

**Semantic Image Recognition Methods  
by Using Deep Learning**

**深層学習を用いた意味的画像認識手法**

**August 2020**

**Hoang Anh DANG**



**Semantic Image Recognition Methods  
by Using Deep Learning**

**深層学習を用いた意味的画像認識手法**

**August 2020**

**Graduate School of Global Information and Telecommunication Studies  
Waseda University**

**Multimedia Representation Research**

**Hoang Anh DANG**



# Contents

<b>Acknowledgments</b>	<b>xv</b>
<b>Abstract</b>	<b>1</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Deep Learning Era . . . . .	5
1.1.1 Artificial Intelligent Milestones . . . . .	6
1.1.2 General Purpose Graphic Processing Unit . . . . .	8
1.1.3 The Renewed Interest in AI . . . . .	8
1.1.4 General Trend . . . . .	9
1.2 Background . . . . .	10
1.2.1 Motivation . . . . .	11
1.2.2 Scalable Vector Graphic AI (SvgAI) . . . . .	11
1.2.3 Street Fashion Semantic Segmentation (SFSS) . . . . .	12
1.3 Objectives . . . . .	13
1.3.1 Scalable Vector Graphic AI (SvgAI) . . . . .	14
1.3.2 Street Fashion Semantic Segmentation (SFSS) . . . . .	14
1.4 Structure of this Thesis . . . . .	14
<b>2 Deep Learning</b>	<b>17</b>
2.1 Milestones of Deep Neural Network (DNN) . . . . .	18
2.1.1 McCulloch and Pitts (MCP) Model . . . . .	18
2.1.2 Hebbian Learning Rule . . . . .	19
2.1.3 Perceptron . . . . .	20
2.1.4 Back Propagation (BP) . . . . .	20
2.1.5 Neocognitron . . . . .	21
2.1.6 Hopfield Network . . . . .	22

2.1.7	Boltzmann Machine (BM)	23
2.1.8	Restricted Boltzmann Machine (RBM)	24
2.1.9	LeNet	25
2.1.10	Other Related Works	26
2.2	Deep Learning Era	26
2.2.1	AlexNet	27
2.2.2	VGG Neural Networks	28
2.2.3	Inception Net	30
2.2.4	Residual Neural Network (ResNet)	31
2.2.5	Multi-GPU Training	32
2.3	Intelligent Agent	35
2.3.1	Q-Learning	36
2.3.2	Policy Gradient	37
2.3.3	Intelligent Agent (IA) Training Loop	38
2.3.4	Experience Memory Replay	39
2.3.5	Exploration and Exploitation	39
2.4	Model-Design v.s. End-to-end network	39
2.4.1	The Edge-Computing Trend	40
<b>3</b>	<b>SvgAI</b>	<b>43</b>
3.1	Vector Image	43
3.2	Previous Works	44
3.3	SVG Editor Environment	45
3.3.1	SVG Construction	45
3.3.2	Action Space	46
3.3.3	State Observation	47
3.3.4	Implementation	47
3.3.4.1	Back-end	47
3.3.4.2	Structure	48
3.3.4.3	Drawing Operation	49
3.4	Model and Algorithms	50
3.4.1	Network Architecture	50
3.4.2	Error and Reward	51
3.4.3	Q-learning and Exploration Policy	52
3.4.4	Policy-Gradient	54
3.5	IA Setting and Evaluation	56
3.5.1	Setting	56

3.5.1.1	Data Set . . . . .	56
3.5.1.2	Episode Termination . . . . .	57
3.5.1.3	Frame Skip . . . . .	57
3.5.1.4	Parameter Update . . . . .	57
3.5.1.5	Parameter Settings . . . . .	57
3.5.2	Evaluation . . . . .	58
3.5.2.1	Performance . . . . .	58
3.5.2.2	Accuracy . . . . .	59
3.5.2.3	SVG Quality . . . . .	61
<b>4</b>	<b>Semantic Segmentation for Street Fashion Photos</b>	<b>65</b>
4.1	Introduction . . . . .	65
4.1.1	Semantic Segmentation . . . . .	65
4.1.2	Semantic Segmentation for Street Fashion Photos . . . . .	66
4.2	Related Works . . . . .	67
4.2.1	Fully Convolutional Neural Network . . . . .	67
4.2.2	PSPNet . . . . .	68
4.2.3	SegNet . . . . .	68
4.2.4	U-Net . . . . .	69
4.2.5	DeepLabv3+ . . . . .	69
4.2.6	Auxiliary Losses . . . . .	70
4.3	Proposals . . . . .	71
4.3.1	Network Design . . . . .	71
4.3.1.1	Problems . . . . .	71
4.3.1.2	Direction . . . . .	72
4.3.1.3	Implementation . . . . .	73
4.3.1.4	Network Structure . . . . .	73
4.3.2	Label Pooling . . . . .	75
4.3.3	Training Objectives . . . . .	76
4.3.3.1	Image Pyramid Loss (IPL) . . . . .	76
4.3.3.2	Segmentation Pyramid Loss (SPL) . . . . .	76
4.3.3.3	Label Pooling Loss (LPL) . . . . .	77
4.4	Setting and Evaluation . . . . .	77
4.4.1	Data Set . . . . .	78
4.4.2	Data Augmentation . . . . .	78
4.4.3	Metrics . . . . .	80
4.4.3.1	mean Intersection over Union (mIoU) . . . . .	80

4.4.3.2	mean Intersection over Union Plus (mIoU+)	80
4.4.4	Ablation Study on Effect of Auxiliary Training Objectives	83
4.4.5	Settings	83
4.4.6	Result	84
<b>5</b>	<b>Conclusion</b>	<b>91</b>
5.1	Scalable Vector Graphic AI (SvgAI)	91
5.2	Street Fashion Semantic Segmentation (SFSS)	92
5.3	Related Issues	92
	<b>Bibliography</b>	<b>94</b>
	<b>List of Academic Achievements</b>	<b>111</b>



# List of Figures

1.1	Notable Achievements of Deep Learning in the 2010s. . . . .	6
1.2	Milestones of Artificial Intelligence . . . . .	7
1.3	Attendances at Large AI Conferences . . . . .	8
1.4	Global top AI tech by number of startups received funding in 2018–2019 . . . .	9
1.5	Relative Position of the Research . . . . .	10
1.6	Research Objectives . . . . .	11
1.7	Research Objectives. . . . .	13
1.8	Structure of This Thesis . . . . .	16
2.1	SIFT Keypoints and Word Stemming. . . . .	17
2.2	Typical Structure of a Feed Forward Artificial Neural Network. . . . .	18
2.3	Structure of Neocognitron . . . . .	22
2.4	Structure of Hopfield Network . . . . .	22
2.5	Structure of Boltzmann Machine . . . . .	24
2.6	Structure of Restricted Boltzmann Machine (RBM) . . . . .	25
2.7	Deep Belief Nets and Deep Boltzmann Machine . . . . .	25
2.8	Structure of LeNet5 . . . . .	26
2.9	Structure of Original AlexNet. . . . .	27
2.10	Structure of AlexNet. . . . .	28
2.11	Visualization of a Convolution Operation. . . . .	28
2.12	Visualization of Pooling Operations. . . . .	29
2.13	Structure of VGG16. . . . .	29
2.14	Inception Modules. . . . .	30
2.15	Factorization Convolution . . . . .	31
2.16	Structure of Inception-v3. . . . .	31
2.17	Structure of ResNet. . . . .	32
2.18	Structure of A Residual Block. . . . .	32

2.19	Distributed Training Race . . . . .	33
2.20	Data Parallelism Multi-GPU Training. . . . .	34
2.21	A General intelligent agent. . . . .	35
2.22	Comparison Between Q-Learning and Policy Gradient. . . . .	37
2.23	The intelligent agent (IA) Training Loop . . . . .	38
2.24	SoCs with Accelerated Computation Capability . . . . .	40
3.1	Raster and Vector Images Comparison. . . . .	43
3.2	Comparison Between SvgAI and Previous Works . . . . .	44
3.3	The proposed framework of SvgAI . . . . .	45
3.4	The process of SVG composition . . . . .	46
3.5	SVG Editor Environment Diagram . . . . .	48
3.6	The architecture of the IA. . . . .	51
3.7	The $\epsilon$ -greedy exploration policy. . . . .	52
3.8	The weighted $\epsilon$ -greedy policy. . . . .	53
3.9	The dual $\epsilon$ -greedy policy. . . . .	53
3.10	Visualize of Target Image Difficulty . . . . .	54
3.11	Examples of Generated Target Images . . . . .	56
3.12	SvgAI Training Performance . . . . .	59
3.13	R2V Result Comparison . . . . .	61
4.1	Samples of VOC2007 data set . . . . .	65
4.2	Deeplabv3+ class semantic segmentation output . . . . .	66
4.3	Samples of Street Fashion Photos . . . . .	67
4.4	Structure of FCN . . . . .	67
4.5	Structure of PSPNet . . . . .	68
4.6	Structure of SegNet . . . . .	68
4.7	Structure of U-Net . . . . .	69
4.8	Structure of DeepLabv3+ . . . . .	70
4.9	Overview of the Proposed Network Structure . . . . .	73
4.10	Label Pooling Visualization . . . . .	75
4.11	Segmentation Results . . . . .	77
4.12	Image Augmentation . . . . .	79
4.13	Illustration of IoU Metric . . . . .	80
4.14	Comparison between mIoU and mIoU+ metric. . . . .	81
4.15	Illustration of IoU+ Metric . . . . .	82
4.16	Network Performance under mIoU metric . . . . .	85
4.17	Network Performance under mIoU+ metric . . . . .	88

4.18 Additional Segmentation Result . . . . .	89
---	----



# List of Tables

2.1	Historical Milestones of Deep Learning . . . . .	19
3.1	List of Actions Supported By The SVG Editor. . . . .	46
3.2	Hyper-Parameter Setting for Experiments. . . . .	58
3.3	IA Performance on Action Set A . . . . .	60
3.4	IA Performance on Action Set B . . . . .	60
3.5	Average SVG Size Comparison . . . . .	61
4.1	Network Parameters . . . . .	72
4.2	ModaNet Data Set Statistic . . . . .	78
4.3	Different Auxiliary Configurations . . . . .	83
4.4	IoU of Individual Classes . . . . .	86
4.5	IoU+ of Individual Classes . . . . .	87



# List of Algorithms

1	Multi-GPU Training with Data Parallelism . . . . .	34
2	Drawing Operation on Cairo's Context . . . . .	49
3	Q-Learning Based Training Algorithm . . . . .	54
4	Policy-Gradient Based Training Algorithm . . . . .	55
5	Adam and AdamW . . . . .	84





# Listings

3.1	Color Parameter Class Abstraction . . . . .	49
3.2	Circle and Square Element Code Excerpt . . . . .	50
3.3	Code Excerpt of Generic Element's Drawing . . . . .	63



# Acknowledgments

Foremost, I would like to express my deepest gratitude to my advisor, Professor Dr. Wataru Kameyama, for his continuous support. My Ph.D. studies would not have been possible without his immense patience, guidance, and motivation.

Secondly, I would like to extend my sincere thanks to Professor Dr. Hiroshi Watanabe and Professor Dr. Jiro Katto for their insightful direction, suggestions, and comments.

I am grateful for the friendliness and hardworking of all the lab members throughout the years. Their work ethic and commitment consistently inspired my interest in research.

I appreciate the professional and devoted assistance from GITS and GITI offices. I also acknowledge the great support from Assistant Professor Dr. Pao Sriprasersuk during my early days at GITS. Their help relieved my burdens to a good extent and allowed me to focus more on my work.

Last but not least, I would like to thank my family for their unconditional trust and endless support.



# Abstract

Artificial intelligence (AI) used to be widely perceived as the field that studies intelligent agent (IA) [1–4]. After the deep learning (DL) breakthrough of AlexNet in 2012 [5], the term AI was frequently used by media, marketers, and academia alike in a much broader sense. In the annual report, the Stanford University’s Institute for Human-Centered Artificial Intelligence (HAI) includes computer vision (CV), Pattern Recognition, Computational Linguistics (CL), Robotics, as the subcategories of AI [6]. In point of fact, AI is generally defined as *intelligence demonstrated by machines*. As such, machine learning and its deep learning subcategory are also parts of IA in computer science.

Since the publication of AlexNet in 2012, DL has been adopted by many branches of science. However, DL sees the most success in the fields of natural language processing (NLP), CV, and IA. In the field of CV, human-level performance in the task of image classification has been achieved by multiple convolutional neural network (CNN) models in 2016. As a result, the ImageNet [7] challenge discontinued in 2017. The end of the ImageNet challenge marks the new chapter in the field. The focus of the research community was shifted to more challenging topics. Some of the notable topics include generative network, semantic segmentation, activity recognition, and visual question answering (VQA). As a whole, the targets of DL researches have become highly semantic.

Significant progress is made among all of the fields in recent years [6]. However, the penetration of AI in everyday life is still limited. Like any other technology, the adoption of DL based AI in the general consumer market lags behind the adoption speed of the industrial market. Not to mention that the adaption speed of the industrial market itself is also lagging behind the research a considerable amount of time.

To be adopted by the industrial and general consumer market, further research must be done to adapt and improve the technology. For example, one of the obstacles that need to be overcome is the hardware limitation. Cloud computing can not satisfy the requirement of real-time processing. And, consumer hardware is still not powerful enough. This limitation leads to the rising trend of edge AI research and development in recent years.

Motivated by the recent success of DL, taking into account the above mentioned low penetration, we aim to enhance the semantic performance and practicality of AI with the two following works:

**Scalable Vector Graphic AI (SvgAI)** [8] is an IA that can draw semantical Scalable Vector Graphics (SVG) images. Instead of storing visual information pixel-by-pixel, SVG image is a document that describes the visual information. Different from natural language, SVG is an Extensible Markup Language (XML) based language. Therefore, SVG is highly semantic and structured hierarchically. Image processing based raster to vector (R2V) converts raster images into SVG without retaining semantic data. With DL, our preliminarily experiments and a related work [9] show the limited performance of networks that shares a similar design to [10] in the task. Rather than tackling the conventional end-to-end model design, we take an alternative approach. We train an IA to perform the task on an SVG editor. The trained IA can create SVG images that are significantly smaller and more accurate compared to the available solutions. Though SvgAI, we show that IA can be used to solve the problem that challenges the conventional end-to-end model. SvgAI is a novel approach to solving the R2V problem.

**Street Fashion Semantic Segmentation (SFSS)** [11] is a light-weight deep neural network (DNN) that performs semantic segmentation on street fashion photos. Introduced just after the release of ModaNet [12], SFSS is a pioneer work on semantic segmentation for street fashion photos. Semantic segmentation can be an essential part of the fashion recommender pipeline. In this work, firstly, we propose a unique and compact DNN design. This network offers state-of-the-art semantic segmentation performance. Furthermore, it requires less computational resources compared to the related works. Secondly, we propose the novel label pooling process, which creates lossless versions of the label in different scales. As the labels for auxiliary training objective, these label pool features significantly improve the context-awareness property of the network.

This thesis is divided into five chapters. [Chapter 3](#) and [Chapter 4](#) are dedicated to the works of SvgAI and SFSS, respectively. Commons to both works are the introduction in [Chapter 1](#), DL background in [Chapter 2](#), and conclusion in [Chapter 5](#). The content of each chapter in this thesis is as follows:

[Chapter 1](#) overviews the history of AI, the contribution of DL into AI development, the achievement of DL, and the impact of this new development on the research trends and business interest. This chapter concludes with the motivation, objectives of the research, and the relative position of this research in the contemporary landscape.

**Chapter 2** introduces the technical background of DL and IA. It includes the milestones of DL and explains popular components of DNN, such as convolutional layer and back-propagation. It also reviews notable DNN models in the DL era, such as AlexNet [5], Inception Net [13–16]. [Section 2.3](#) describes IA and the two popular algorithms to train an IA, including Q-Learning and policy gradient. Important concepts related to the training process, including experience memory replay (EMR) and exploration policy, are also addressed in this chapter.

**Chapter 3** is dedicated to the work of SvgAI. This chapter starts with an introduction to the R2V problem. The introduction also includes reviews on previous works, and the challenges need to be resolved. Different from the previous works on the topic, we proposed a novel framework for R2V.

In our new framework, an IA is trained to draw vector images using an SVG editor. In order to train the IA, a complete training environment is needed. The design and implementation of our SVG editor environment are explained in [Section 3.3.4](#).

The latter half of the chapter describes the experimental setting and evaluation result of SvgAI using both deep Q-Learning and gradient-policy. Dual  $\epsilon$ -greedy exploration strategy and a unique training strategy are proposed to overcome the difficulties. The chapter is concluded by a comparison between SVG images produced by SvgAI with popular free and commercial R2V software.

**Chapter 4** is dedicated to the work of SFSS. This chapter begins with the introduction to semantic segmentation. Then, we review related works on the topic, including SegNet [17], DeepLabv3+ [18], and PSPNet [19].

Different from the previous works, we focus on the semantic segmentation task for fashion apparel. To produce the most efficient model for the task, we propose two novel contributions. They are: 1) a high-performance semantic segmentation DNN that follows the encoder-decoder structure, and 2) the 2D max-pooling-based scaling operation.

We train and evaluate our proposed network using the ModaNet data set. To better evaluate the network performance, the Intersection over Union Plus (IoU+) metric is also proposed. This metric is taking noise into account for better evaluation. An ablation study is conducted to analyze the effect of different auxiliary training losses.

**Chapter 5** concludes this dissertation with possible directions for future works.





## CHAPTER 1

# Introduction

This thesis describes semantic processing for object shape processing using deep learning. While the semantic object shape processing goal is broad, we especially emphasize on object shape description and recognition via two works below.

**Scalable Vector Graphic AI (SvgAI)** In this work, we present an intelligent agent (IA) that can draw Scalable Vector Graphics (SVG) image. This work is a pioneer work in using IA in raster to vector (R2V) problem.

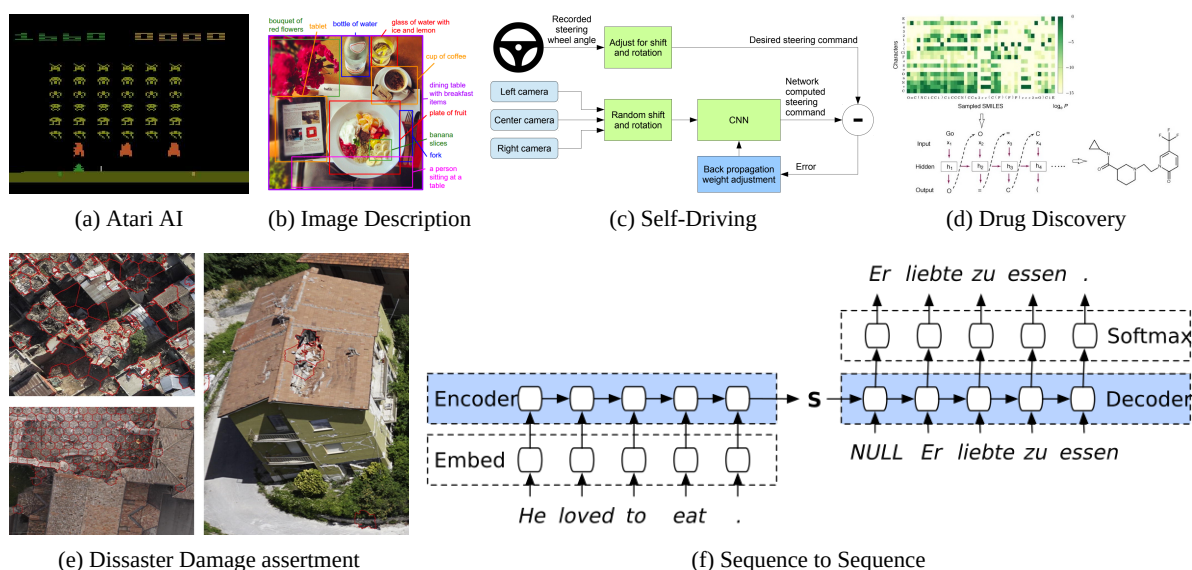
**Street Fashion Semantic Segmentation (SFSS)** In this work, we propose a state-of-the-art, end-to-end deep neural network (DNN) for semantic segmentation on street fashion photo. We also propose a constrained training process using features generated by a novel label pooling process.

In this chapter, we present the background of the research, which includes the overview of deep learning (DL) and artificial intelligence (AI) development in recent years. We then discuss the current direction of DL research as a whole. Finally, we present the background and research objective of each work.

## 1.1 Deep Learning Era

The time after 2010 is usually depicted as the DL era in media and literature. Conceivably, the name is originated by the absolute dominant in performance and popularity of deep learning in all branches of research during this time. [Figure 1.1](#) show some of the achievements of DL in the 2010s. Nevertheless, the figure is not an exhaustive list. Other remarkable achievements are: word2vec [20], generative adversarial nets (GAN) [21, 22], style transfer [23–26], to name a few.

In 2012, Krizhevsky et al. released AlexNet [5] and win the first prize in ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012. AlexNet is a CNN with five convolutional



**FIGURE 1.1** Notable Achievements of Deep Learning in the 2010s. **(a)** IA that play Atari games well above the skill of humans created by DeepMind. The IA was trained using AI Gym [27]. **(b)** Automatic image description made possible by cross-training language and visual model [10]. **(c)** End-to-end neural network training for self-driving car [28]. **(d)** Drug discovery by using RNN [29–31] to generate molecular structure [32]. **(e)** Disaster damage assessment using aerial image using CNN and 3D point cloud [33]. **(f)** Sequence-to-sequence language model being used for translation [34].

Figures taken from corresponding cited references.

layers, followed by a two-layer classifier. It reduces the error rate by further 10% compared to the best model in ILSVRC 2011. The astounding result resurrects the interest in DL. This new wave of interest started the DL era.

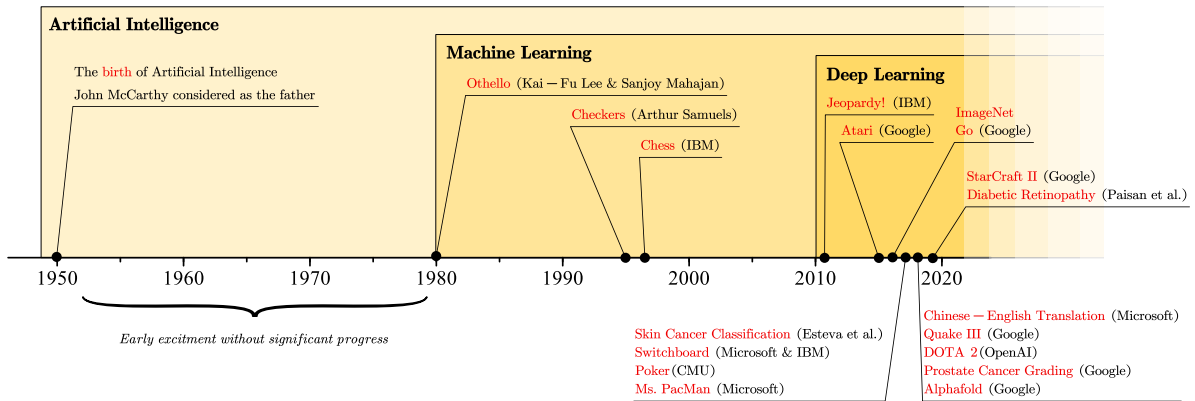
### 1.1.1 Artificial Intelligent Milestones

Figure 1.2 shows the milestones of AI developments and the hierarchical relation between AI, machine learning (ML), and DL. The progress of AI development is marked by the moment when AI achieved humans performance in specific tasks.

As shown in the figure, this progress has been rapid in recent years. The number of milestones in 2018 alone surpasses the number of milestones in 30 years from 1980 until 2010 [6]. Moreover, all of these milestones were achieved using DL. Following is the list of notable milestones.

**Othello** The program named BILL [35] that can play the board game named "Othello" was created by Lee and Mahajan. In 1989, this program beat the highest-ranked US player, Brian Rose, by the score 56—8.

**Checkers** Chinook [36], a program that can play checkers built by Schaeffer et al., beat the



**FIGURE 1.2** Milestones of Artificial Intelligence

world champion in 1995.

**Chess** In 1997, IBM’s DeepBlue [37] system beat the chess champion, Gary Kasparov.

**Jeopardy!** In 2011, IBM’s Watson computer system defeated the former winners of Jeopardy! TV show and won the first prize.

**Atari Games** In 2015, Mnih et al. at Google DeepMind released a DNN that can play Atari Games at human-level performance [38].

**ImageNet** In 2016, the error rates of various CNN models in the ILSVRC challenge were less than 3%. This error rate is even lower than the 5% error rate of humans.

**AlphaGo** In 2016, the IA named AlphaGo [39], developed by Silver et al. at Google DeepMind, beat the world’s best Go player.

**Skin Cancer Classifier** In 2017, Esteva et al. described an AI system that can classifying skin cancer images in the same level with a dermatologist.

**Speech Recognition** In 2017, research groups from Microsoft [41] and IBM [42] both achieved human-level speech recognition for switchboard applications.

**Poker** In 2017, both Libratus [43] made by MCU and DeepStack [44] made by the University of Alberta achieved poker playing skill on par with professionals.

**Chinese-English Translation** In 2018, Microsoft achieved human-level quality in Chinese to English translation [45].

**Alphastar** was developed by Google’s DeepMind [46] in 2019. It was reported to defeat professional players in the StarCraft II game.

### 1.1.2 General Purpose Graphic Processing Unit

The first modern CNN, LeNet [47], was released in 1989. However, due to the high demand for computational power, they are forgotten in favor of faster, model-design techniques.

Nvidia popularized the use of graphic processing unit (GPU) for general-purpose (i.e., general purpose graphic processing unit (GPGPU)) with the first version of Nvidia CUDA released in 2007. Computational power-wise, GPGPU is vastly cheaper compared to CPU. As in spring 2020, a 16 cores CPU Ryzen 9–3950X and an Nvidia RTX-2080 Super are priced the same at around 750USD. However, the RTX-2080 Super has theoretical performance at 11.1 TFLOPS, and the 3950X performance is 0.97 TFLOPS.

The cheap computational cost of GPGPU allows training the DNN cheaply and effectively. DNN and deep convolutional neural network (DCNN), with many more layers, have an exceptional ability to generalize data.

### 1.1.3 The Renewed Interest in AI

The success of CNN rekindled the interest in AI from media, academia, and business, This wave of interest in DL is the third wave since the birth of AI. Figure 1.2 shows the milestones of AI development. The three waves of interest of AI can be observed in this figure. The first wave started with the birth of AI itself. The second wave started with ML. ML distinguished from the previous works by the ability to learn of a model without explicit coding. The third wave started the current DL era.

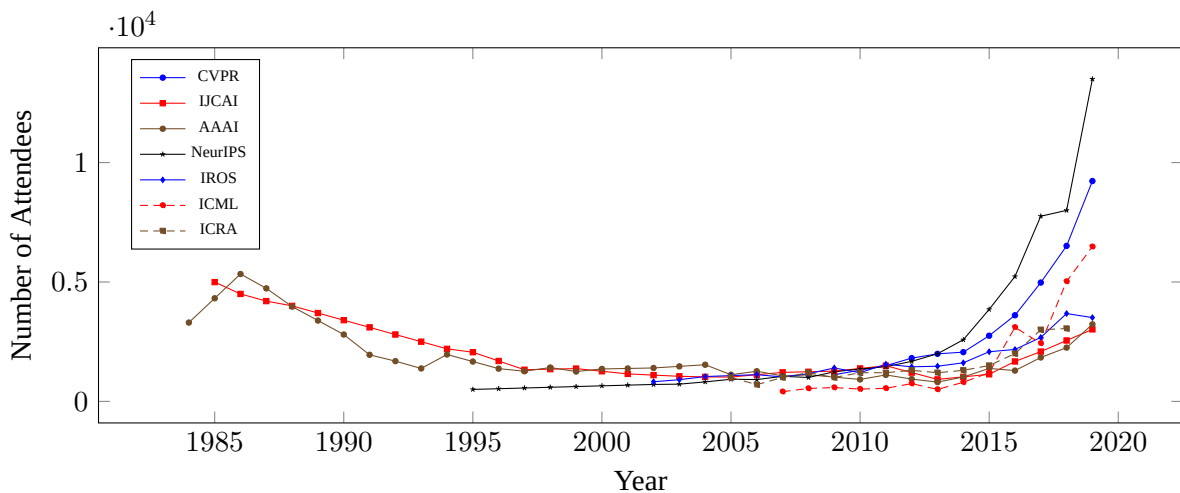
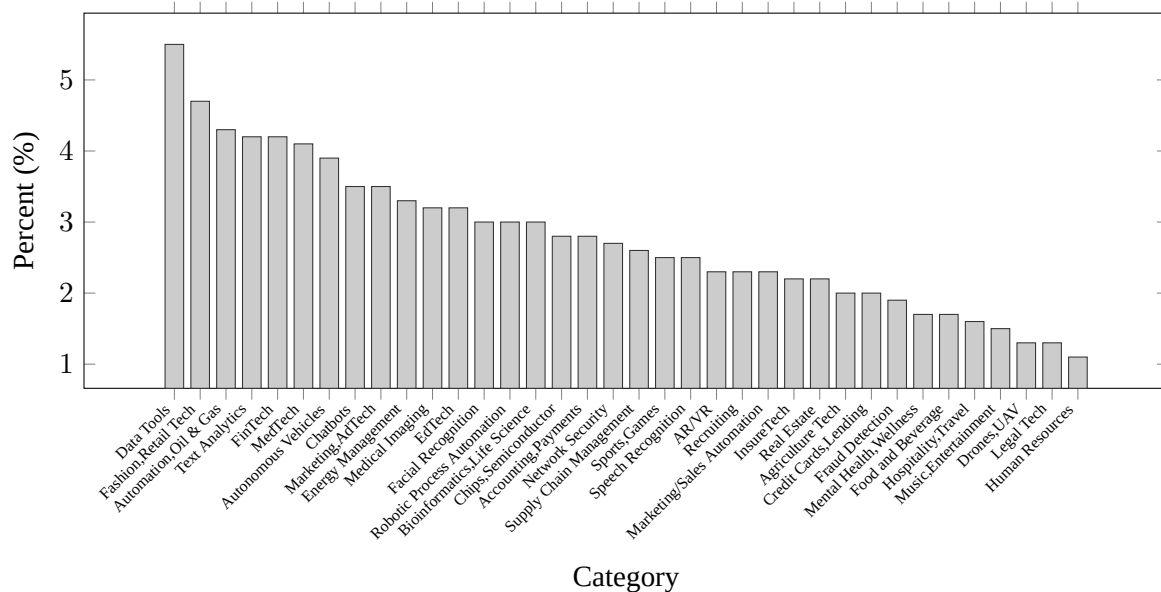


FIGURE 1.3 Attendances at Large AI Conferences (1984–2019) [6]

Figure 1.3 shows the number of attendees at large AI conferences from 1984 to 2019. We can observe the second wave of interest faded gradually every year after 1986. After 2012, the number of attendees in these large conferences grown exponentially. It clearly reflect the

unprecedented level of excitement of the research community towards DL as well as AI in general.

The new excitement in AI leads to an equally impressive amount of investment into new AI tech. According to [6], in 2010, the total amount of investment in new AI startups was only \$1.3B. However, in 2018 alone, this number was \$40.4B, more than thirty times compared to the investment made in 2010. Thus, the average annual growth rate of investment in AI startups was 48%.



**FIGURE 1.4** Global top AI tech by number of startups received funding in 2018–2019 [6]

Figure 1.4 shows the global top category of AI technology by the number of startups that received funding in 2018–2019. Interestingly, the top categories are Data Tool (5.5%), Fashion and Retail Tech (4.7%), and Industrial Automation (4.3%). Even though many different factors drive the research trend, we believe that business investment is one of the most significant influences. Therefore, we expect these aforementioned top categories will continue to be the prominent trends for AI research.

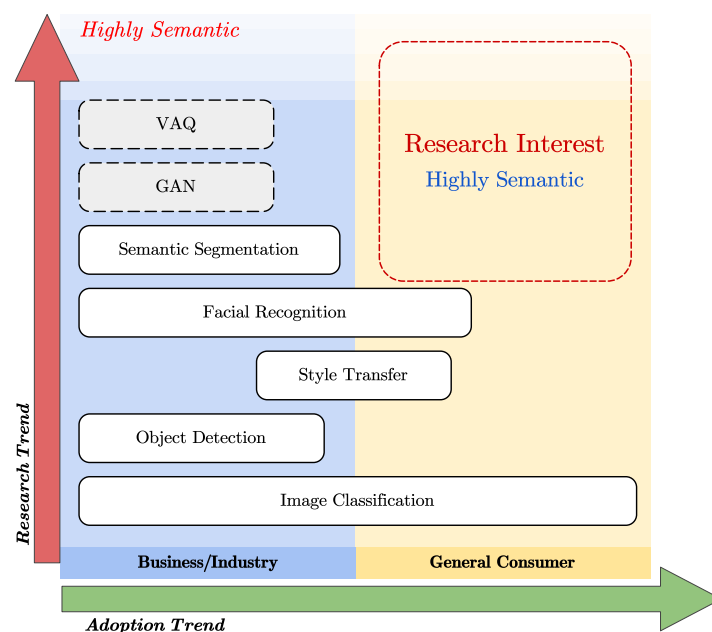
### 1.1.4 General Trend

After various CNN models achieved the human-level in image classification task in 2016, the ImageNet [7] challenge discontinued in 2017. The end of the ImageNet challenge marks the new chapter in AI development. The focus of the research community shifted to more challenging and more semantical tasks. This shift of focus can be seen through the milestones of AI (Section 1.1.1). Most of the milestones after 2017 focus on IAs, natural language processing (NLP), and medical image semantic segmentation.

This trend is still ongoing in 2020, as the research community continues to try to achieve human performance in new tasks [6]. However, the penetration of AI in everyday life is still limited. Like any other technology, the adoption of DL based AI in the general consumer market lags behind the adoption speed of the industrial market. Not to mention that the adaption speed of the industrial market itself is also lagging behind the research a considerable amount of time.

To be adopted by the industrial and general consumer market, further research needs to be done to adapt and improve the technology. For example, one of the obstacles that need to be overcome is the hardware limitation. Cloud computing can not satisfy the requirement of real-time processing. And, consumer hardware is still not powerful enough. This limitation leads to the rising trend of edge AI research and development in recent years.

Another reason for the low impact of AI is because of data interpretation. Modern DNN heavily relies on convolutional layers. As a result, modern DNN performs best on spatial, sequence, or time-series data. However, a large amount of data we generated is not in such a format. Some examples of such kind of data include social connections, street maps, electric grid.



**FIGURE 1.5** Relative Position of the Research

## 1.2 Background

In this section, we introduce the background of our works and then conclude the chapter by our research objective.

### 1.2.1 Motivation

Figure 1.5 shows the relative position of the research concerning the trends of AI research and market adoption. In general, the market can be divided into two segments, business and general consumer. Like any other technology, the adoption of the market is usually lagging behind the research progress. This lag is also happening in the adoption trend. The business is the first segment to adopt new technologies. The general consumer market usually adopts a technology once it robust enough.

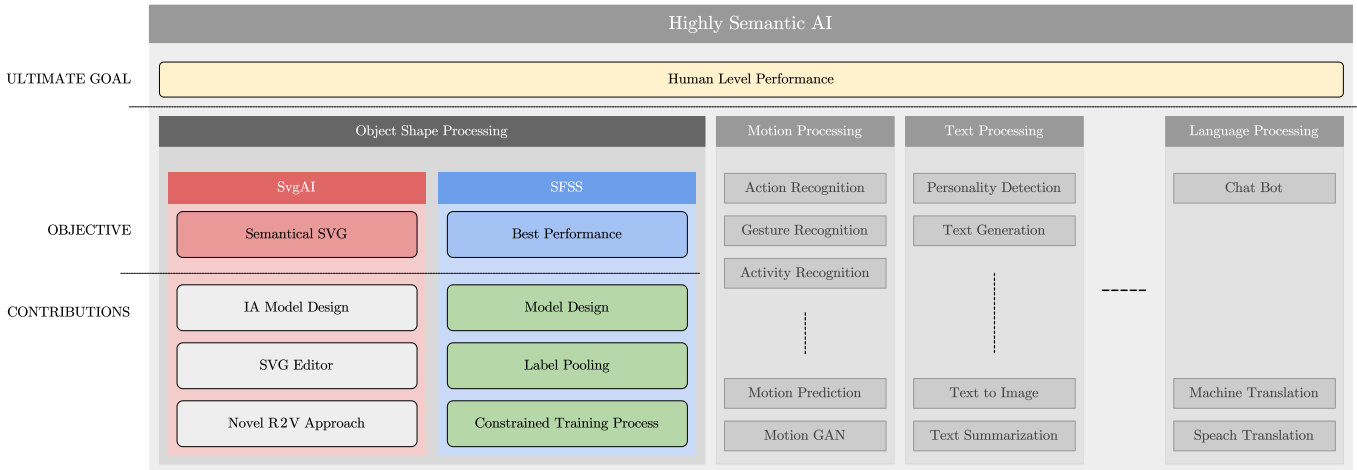


FIGURE 1.6 Research Objectives

Our research interest places on the top right corner of Figure 1.5. In this region, technology not only needs to be highly semantic but also needs to be practical. We classify works in this region based on their semantic category, as in Figure 1.6. As shown in the figure, our two works of SvgaI and SFSS belongs to the object shape processing category. There are notable topics in other categories in this area, such as motion GAN in motion processing category, text summarization in text processing category, and speech translation in language processing, to name a few.

### 1.2.2 Scalable Vector Graphic AI (SvgaI)

Despite being a mature branch of research, image processing based R2V conversion is not yet reliable [48]. Major problems include difficulties of color quantization, aliasing effects, shift, superposition effects, and miss-identification of texture and text [49].

There are numerous works to try to solve the above-mentioned problems. For example, Kansal and Kumar propose a framework to reproduce the linear filled gradient [50]. Vector representation of halftone dots in binary images is presented by Kawamura et al. [51]. However, the human operator still needs to identify the type of problems and perform appropriate

parameter tweaking. Thus, for example, well-known conversion tools such as Potrace [52] still require humans intervention to achieve desirable results [53].

As a result, conventional R2V conversion usually produces SVG images using path elements exclusively because this element is flexible and can be used to form any shape. It not only inflates the size of the SVG but also poses a high demand for computational resources for image rendering. Furthermore, using fundamental shape elements, such as rectangle, circle, and arc, not only requires a higher level of visual understanding but also coming with none trivial challenges.

The topic of R2V, unfortunately, is not well investigated in deep learning. Vector image fundamentally is a text document based on Extensible Markup Language (XML) [54]. Though RNN sees much success with translation and text writing. XML is a highly hierarchical structured text document with open and closing tags. For this reason, there is no natural method for a neural network (NN) to output XML documents directly. Beltramelli proposes Pix2Code [9], an end-to-end DNN that generates XML based graphical user interface (GUI) code from mock-up images. Even though similar to [10] in general design, the visual model used in Pix2Code is a plain CNN block while the language model is handled by a Long-Short Term Memory (LSTM) network [55] block. Another LSTM block is used to decode the network's output into code tokens. This work can be understood as a rigid version of automated image annotation. However, the model is not flexible because the visual presentations of all the GUI elements in this work are predefined templates. Thus, it only works with GUI images based on the fixed templates.

### 1.2.3 Street Fashion Semantic Segmentation (SFSS)

Semantic segmentation has been a challenge in the field of computer vision (CV). Classic object detection and classification requires only bounding boxes and classification of the object. Semantic segmentation needs each pixel in the input image to be assigned to a class of objects. Figure 1.7 shows examples of inputs and corresponding ground-truth labels in semantic segmentation problem for street fashion photos.

Before the deep learning era, the state-of-the-art works have been based on Texton Forest [56] and conditional random field (CRF) [57]. CRF is still being used as a post-process method to refine the segmentation output [19, 58–62].

Early DL works on this topic mostly adopt the straight network design. Fully convolutional neural network (FCN) [58] has laid the foundation for applying CNN into dense segmentation. It can be implemented on top of the ever-proposed classification models such as GoogLeNet [13], VGG [63], and ResNet [64].

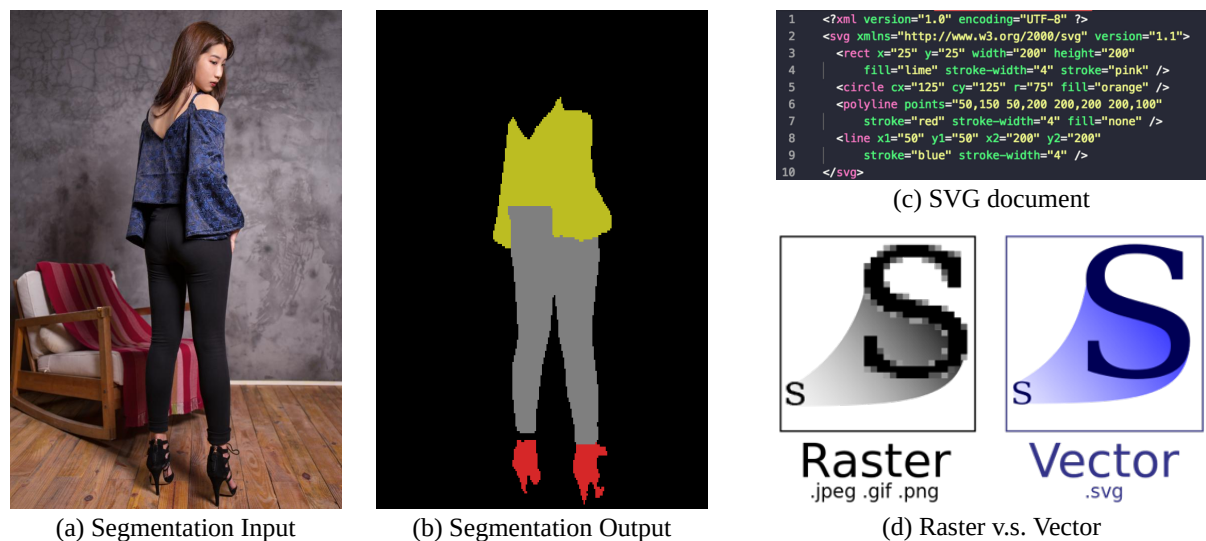
PSPNet [19] introduces a spatial pyramid pooling scheme, which results in better context-



awareness in the final result. In this pyramid pooling scheme, features maps from different layers of the base network are resized and concatenated. The concatenated feature map is then used as input for a point-wise CNN to produce segmentation results.

Later works on the topic mostly utilize the encoder-decoder structure. Models following this approach is usually yielding better performance. Popular models in this category include SegNet [17] and U-Net [65]. In [59], Yu and Koltun propose both dilated CNN for semantic segmentation and a reference network design. Dilated CNN allows the deeper layers of the network to capture the context without losing resolution. The main drawback of this design is the high demand for computational resources because the feature map is rarely down-sampled. DeepLabv3+ [18] combines all of the above approaches and achieves state-of-the-art performance in many benchmarks.

MSCOCO [66], CityScapes [67], and ADE20K [68] are popular datasets for training and benchmarking semantic segmentation works.



**FIGURE 1.7** Illustration of Research Objectives. In SFSS task, we design a light-weight model that accepts photo as an input (a) and output segmentation result as in (b). In R2V task, we train an IA to use SVG editor to convert raster image to SVG image as in (d). SVG image is an XML based document as shown in (c).

Image (d) courtesy of Wikipedia.

### 1.3 Objectives

Both of our works align with the trajectory of CV research: to improve network performance, reaching closer to human-performance. At the same time, we would like to explore the capability of DNN in new domains that could potentially benefit the everyday user. Figure 1.6 illustrates the research objectives and mote notable contributions of each work.

### 1.3.1 Scalable Vector Graphic AI (SvgAI)

In the work of SvgAI, inspired by the work of Karpathy and Fei-Fei on image annotation [10], we ought to study a DNN that can convert raster images to SVG format. Most importantly, this new system must retain the semantic structure of the image and reflect it into the SVG output. In [10], the authors utilized an end-to-end cross-modal design DNN. However, preliminary experiments on the end-to-end model showed unfavorable results. We took an alternative route and proposed a novel framework for R2V that uses IA to compose SVG. Ultimately, this new system must produce lighter, more semantical SVG compared to the available solutions.

### 1.3.2 Street Fashion Semantic Segmentation (SFSS)

With the newly released ModaNet [12], we tasked to propose a light-weight state-of-the-art model for street fashion photo semantic segmentation. This model is an end-to-end network that can correctly identify the categories of apparels on the subject. It is expected to cost not only a less computational resource but also achieve higher segmentation performance.

## 1.4 Structure of this Thesis

This thesis is organized into five chapters in which this is the first chapter. In this chapter, we provided an overview of the DL era. Then, we described the research background and objective of both SvgAI and SFSS. [Figure 1.8](#) illustrates the structure as well as the content overview of this thesis. The following chapters are organized as follows:

**Chapter 2** introduces the technical background of DL and IA. It includes the milestones of DL and explains popular components of DNN, such as convolutional layer and back-propagation. It also reviews notable DNN models in the DL era, such as AlexNet [5], Inception Net [13–16]. [Section 2.3](#) describes IA and the two popular algorithms to train an IA, including Q-Learning and policy gradient. Important concepts related to the training process, including experience memory replay (EMR) and exploration policy, are also addressed in this chapter.

**Chapter 3** is dedicated to the work of SvgAI. This chapter starts with an introduction to the R2V problem. The introduction also includes reviews on previous works, and the challenges need to be resolved. Different from the previous works on the topic, we proposed a novel framework for R2V.

In our new framework, an IA is trained to draw vector images using an SVG editor. In order to train the IA, a complete training environment is needed. The design and implementation of our SVG editor environment are explained in [Section 3.3.4](#).

The latter half of the chapter describes the experimental setting and evaluation result of SvgAI using both deep Q-Learning and gradient-policy. Dual  $\epsilon$ -greedy exploration strategy and a unique training strategy are proposed to overcome the difficulties. The chapter is concluded by a comparison between SVG images produced by SvgAI with popular free and commercial R2V software.

**Chapter 4** is dedicated to the work of SFSS. This chapter begins with the introduction to semantic segmentation. Then, we review related works on the topic, including SegNet [17], DeepLabv3+ [18], and PSPNet [19].

Different from the previous works, we focus on the semantic segmentation task for fashion apparel. To produce the most efficient model for the task, we propose two novel contributions. They are: 1) a high-performance semantic segmentation DNN that follows the encoder-decoder structure, and 2) the 2D max-pooling-based scaling operation.

We train and evaluate our proposed network using the ModaNet data set. To better evaluate the network performance, the Intersection over Union Plus (IoU+) metric is also proposed. This metric is taking noise into account for better evaluation. An ablation study is conducted to analyze the effect of different auxiliary training losses.

**Chapter 5** concludes this dissertation with possible directions for future works.

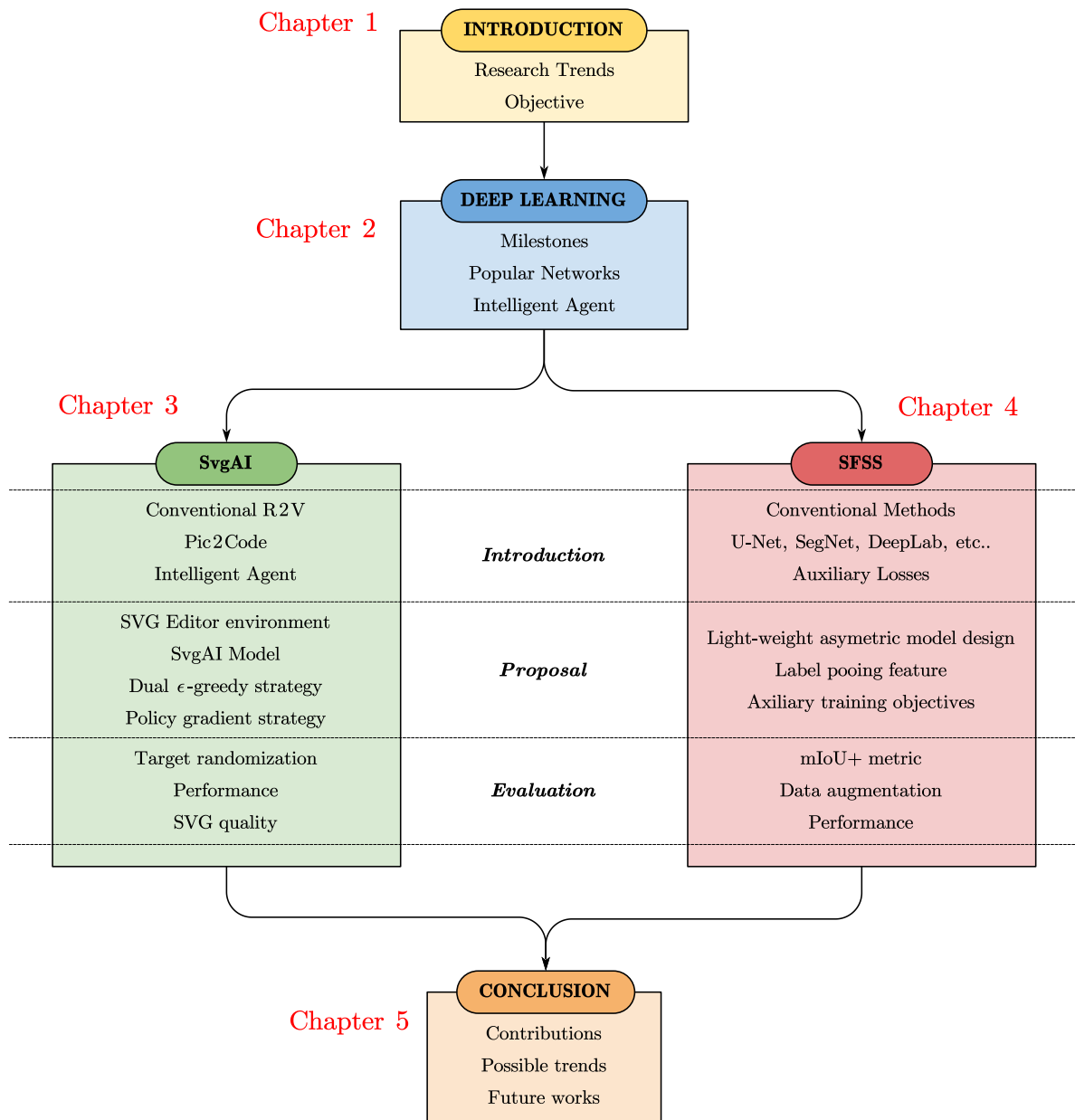
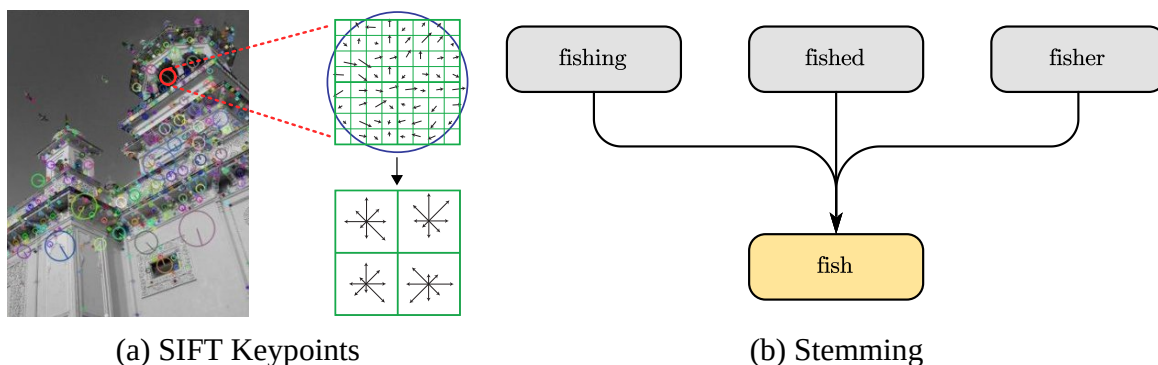


FIGURE 1.8 Structure of This Thesis

## CHAPTER 2

# Deep Learning

The majority of computer vision (CV) models, or machine learning (ML) models in general, consist of two parts. The first part (i.e., head) is feature extraction, and the second part (i.e., tail) is a classifier or a regressor. In conventional Feature extraction usually requires a high degree of domain knowledge to be designed. In conventional CV, popular extractors are SIFT [69], SURF [70], ORB [71]. Popular classifiers and regressors are SVM [72], Random Forest [73, 74], Bayesian network, and Logistic model. Even with artificial neural networks (ANN), the two parts structure is holding true.

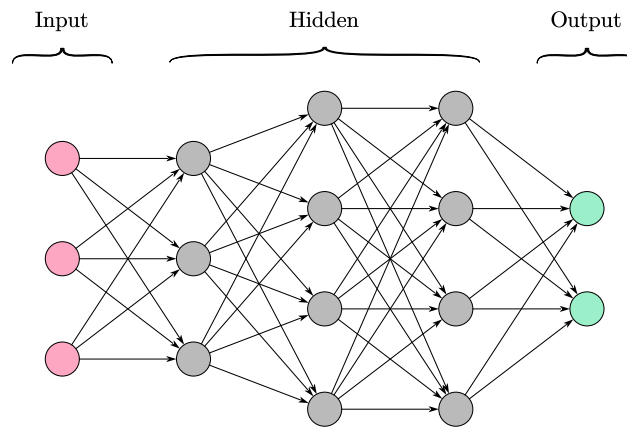


**FIGURE 2.1** SIFT [69] Keypoints and Word Stemming.

Raster image in (a) courtesy of OpenCV.

Figure 2.1 shows the scale-invariant feature transform (SIFT) [69] keypoints and word stemming. Stemming is the process of reducing all words into their root form. This process is nearly mandatory in conventional natural language processing (NLP). SIFT keypoints are popular in CV research. Applications such as image-stitching are still robust and still being used.

An ANN is an interconnected group of artificial neuron nodes. The most simple form of an artificial neuron node receives multiple inputs to produce an output. Mathematically, the output of a neuron can be described as a non-linear transformation  $y = \text{activation}(Wx + b)$ . Multiple nodes that have the same input form a layer. It is called a fully connected (FC) layer because every output is connected to all of the inputs.



**FIGURE 2.2** Typical Structure of a Feed Forward Artificial Neural Network.

The most common form of ANN is the feed-forward neural network (FFNN). FFNN usually comprises of several layers of neurons, as shown in [Figure 2.2](#). As shown in the figure, an ANN can receive input and produce output directly. Thus, forming an end-to-end model in which the boundary between head and tail in becomes blurry.

An ANN with a large number of layers of neural network forms a deep neural network (DNN). Though, there is no official number of layers for an ANN to be defined as a DNN. LeNet5, with five layers of neurons, is considered as a DNN. Thus, it is sufficient to classify an ANN with more than five layers as a DNN based on that standard. The study of DNNs is called deep learning (DL).

## 2.1 Milestones of Deep Neural Network (DNN)

The 2010s have witnessed many technological advancements enabled by DL. The history of DL and neural network (NN) can be traced back to several decades. However, not until early of the 2010s, DNNs had achieved notable success in the field of CV. This success promoted the use of DNN into many different fields, as we have seen nowadays. In this section, we introduce a brief history of DNN.

Conceivably, it not easy to trace the lineage of DNN development correctly. However, there are typical works that frequently mentioned as such as shown in [Table 2.1](#).

### 2.1.1 McCulloch and Pitts (MCP) Model

McCulloch and Pitts (MCP) [75] model proposed by McCulloch and Pitts in 1943. This model was designed for the electrical circuit. It is usually mentioned as McCulloch Pitts Neurons in literature. It also is considered the first artificial neurons in many articles. The model

**TABLE 2.1** Historical Milestones of Deep Learning

Year	Contributor	Contribution	Note
1943	McCulloch and Pitts	MCP model [75]	designed for electrical circuits
1949	Hebb and Hebb	Hebbian learning rule [76]	unstable is the main drawback [77]
1958	Rosenblatt	Perceptron [78]	origin of a modern dense NN cell
1974	Werbos	back propagation (BP) [79]	it is not gradient descent (GD)
1980	Fukushima	Neocognitron [80]	inspiration of the modern CNN
1982	Hopfield	Hopfield Network [81]	i.e. "fully connected" network
1985	Ackley et al.	Boltzmann Machine (BM) [82]	Hopfield network with hidden units
1986	Smolensky	Restricted BM [83]	originally known as Hormonium
1990	LeCun et al.	LeNet [84]	modern CNN

mathematically described as follows.

$$y = \begin{cases} 1, & \sum_i w_i x_i \geq \theta \quad \text{AND} \quad z_j = 0, \forall j \\ 0, & \text{otherwise} \end{cases} \quad (2.1)$$

Where  $y$  stands for output,  $x_i$  stands for input,  $w_i$  stands for the corresponding weights, and  $z_j$  stands for inhibitory input. Hence, the model resembling a modern artificial neuron unit with a binary step as an activation function. However, the MCP unit has inhibitory input, which is the main difference from an ordinary modern artificial neuron. Also, MCP does not have a bias term compared to a modern neuron unit. Because this model is proposed for the electrical circuit, the weight of the network has to be predetermined based on the application.

### 2.1.2 Hebbian Learning Rule

Hebbian learning rule [76] has proposed by Hebb and Hebb in 1949. Consider a typical linear neural  $y = \sum_i w_i x_i$ . The Hebbian learning rule is usually generalized as follows.

$$\Delta w_i = \eta x_i y \quad (2.2)$$

Where  $\Delta w_i$  stands for the change of synaptic weights  $w_i$  of Neuron  $i$ , of which the input signal is  $x_i$ ,  $y$  denotes the post-synaptic response, and  $\eta$  denotes learning rate. However, being one of the first learning theories, Hebbian learning rule is suffered from the unstableness problem [77]. This lead to many later works adopted other theories such as Oja rules [85] or Generalized Hebbian Algorithm (GHA) [86].

### 2.1.3 Perceptron

Perceptron [78] is an artificial neuron unit proposed by Rosenblatt in 1958. A perceptron is mathematically described as follows.

$$y = \begin{cases} 1, & \sum_i w_i x_i + b > 0 \\ 0, & \text{otherwise} \end{cases} \quad (2.3)$$

Where  $w_i$ ,  $x_i$  are the weight and input of neuron  $i$ , and  $b$  is the decision threshold. The only difference between a perception and a modern artificial neuron unit is the activation function. Modern artificial neuron units have numerous different activation functions. However, perceptron only uses the binary step as the activation function.

A shallow network comprised of only one hidden layer of perceptron units can describe any bounded continuous function. This property is called universal approximation [87]. However, universal approximation with shallow networks comes at the price of an exponential number of neuron units. Thus, demanding an unpractical amount of computational resource.

Compared to the MCP model, the perceptron has a closer resemblance with a modern artificial neuron unit. Therefore, the perceptron is sometimes mentioned in literature as the first modern artificial neuron unit. Perceptron eventually evolved into the modern artificial neuron unit, starting with the use of sigmoid as activation function [88].

### 2.1.4 Back Propagation (BP)

First introduced by Werbos in his Ph.D. thesis in 1974, as its name suggested, BP is the process of propagating error back to each original value. The process is done by utilizing the chain rule to compute partial derivative (i.e., gradient) of a variable. For example, consider the following function. <sup>1</sup>

$$f(x, y, z) = (x + y)z \quad (2.4)$$

Given  $q(x, y) = x + y$ , we have  $f(x, y, z) = f(q, z)$  and following partial derivatives:

$$f(q, z) = qz \quad \implies \quad \frac{\partial f(q, z)}{\partial q} = z \quad , \quad \frac{\partial f(q, z)}{\partial z} = q \quad (2.5)$$

$$q(x, y) = x + y \quad \implies \quad \frac{\partial q(x, y)}{\partial x} = 1 \quad , \quad \frac{\partial q(x, y)}{\partial y} = 1 \quad (2.6)$$

With  $q$  already computed, the gradient of  $f$  in respect to  $z$  (i.e.,  $\frac{\partial f}{\partial z}$ ) is obtained as in [Equa-](#)

---

<sup>1</sup>The example in this subsection is adapted and carried over from an online tutorial [89] written by Karpathy.



tion (2.5). However, the gradient of  $f$  with respect to  $x$  and  $y$  need to be computed following the chain rule as follows.

$$\frac{\partial f(q, z)}{\partial x} = \frac{\partial q(x, y)}{\partial x} \frac{\partial f(q, z)}{\partial q} = 1z = z \quad (2.7)$$

$$\frac{\partial f(q, z)}{\partial y} = \frac{\partial q(x, y)}{\partial y} \frac{\partial f(q, z)}{\partial q} = 1z = z \quad (2.8)$$

In the end, by multiplying the gradient of each variable by the error of the function  $f$ , we obtained the 'error' of each variable  $x$ ,  $y$ , and  $z$ .

Note that the term BP and GD are two different terms. GD [90] is the process of updating the weight of the network so that the error is descent in hyperspace. In other words, BP and GD are the two processes that make up a single training step. Stochastic gradient descent (SGD) [91, 92] is the most popular variant of GD algorithms being used contemporary.

Besides the work of Werbos, there are reports of multiple independence discover of BP around the same time [93–97]. The term BP and its use for NN are coined and popularized by Rumelhart et al. [98, 99].

The classic SGD algorithm is difficult and slow to train. Even with batch normalization, the learning rate has to be set cautiously to achieve optimal training results. A series of adaptive optimizers have been proposed to mitigate the complication of the training process. Popular algorithms includes Momentum [100], Nesterov accelerated gradient (NAG) [101], Adagrad [102], AdaDelta [103], RMSProp [104], Adam [105], Nadam [106], AMSGrad [107]. These optimizers monitor the gradient of each parameter and tweak the learning rate of the individual parameter accordingly. Thus, they are not as sensitive to global learning rate changes compared to vanilla SGD.

### 2.1.5 Neocognitron

Neocognitron is a pioneer work on ANN published in 1980 by Fukushima. In this work, the author proposed a hierarchical, multilayer ANN that strikingly resemblance the nowadays convolutional neural networks (CNNs).

The structure of Neocognitron is shown in Figure 2.3. As shown in the figure, Neocognitron mainly consists of two different kinds of layers. S-layer as a feature extractor and C-layer as structured connections to organize the extracted features.

Fukushima conducted experiments on text recognition and reported positive results. However, given the constrain of computational resources at the time, the full simulation was not feasible. Neocognitron, later on, became an inspiration for subsequence works on CNNs such as LeNet [84]. Due to high computational demands, Neocognitron or CNN did not attract much

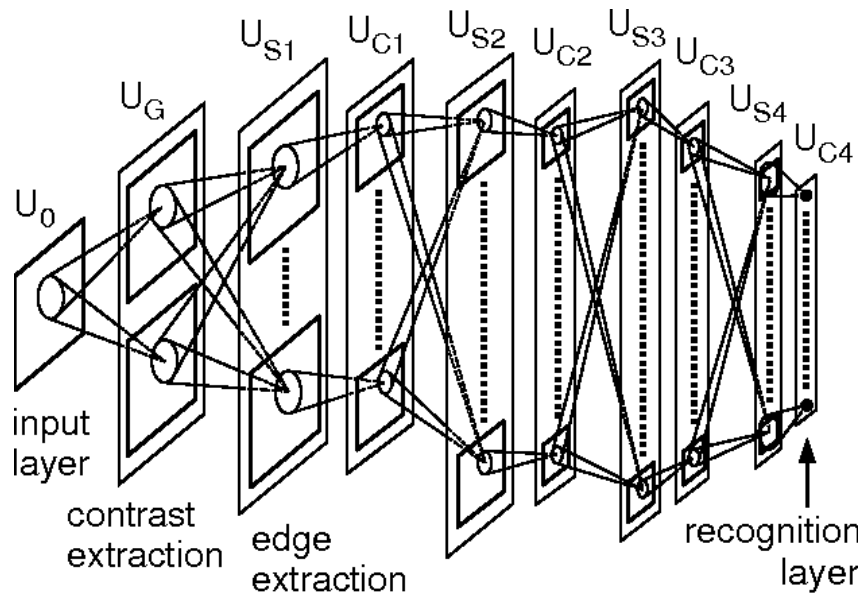


FIGURE 2.3 Structure of Neocognitron [80].

attention until the computational problem solved with general purpose graphic processing unit (GPGPU).

### 2.1.6 Hopfield Network

Hopfield network is proposed in 1982 by Hopfield. It is a form of recurrent neural network (RNN). It is designed to work with binary input and output. Hopfield network also called Fully Connected Network and widely recognized because of its content-addressable memory (CAM) property.

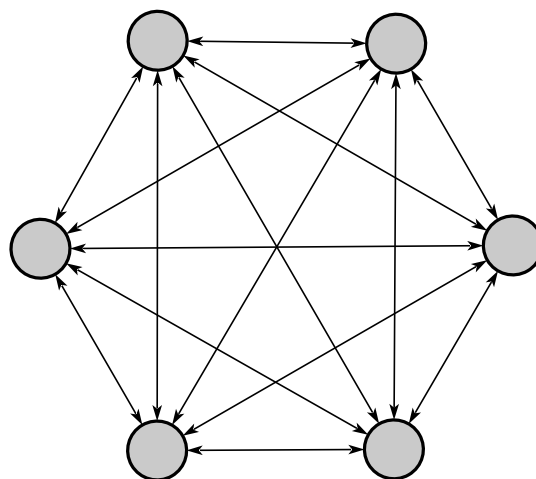


FIGURE 2.4 Structure of Hopfield Network [81].

Figure 2.4 shows the structure of the six nodes Hopfield network. Each node served as an input and an output. Formally, Hopfield network can be defined as a graph  $G = \langle W, S \rangle$ . Where,  $W$  is the set of edge weights and  $S$  is the set of vertexes. Hopfield networks typically has two restrictions:  $w_{ii} = 0, \forall i$  and  $w_{ij} = w_{ji}, \forall i, j$ . Where  $w_{ij}$  is the edge weight between vertex  $s_i$  and  $s_j$ . Thus,  $W$  is symmetric, and a neuron (i.e., vertex) is not connected to itself.

To store a pattern, a Hopfield network is trained using Hebbian learning  $w_{ji} = y_j y_i$ . In short, it is setting the edge value to the product of the two values (vertex) it is trying to store. The inference process of the Hopfield network is done by multiple iterations. In each iteration, the network is updated as follows.

$$s_i \leftarrow \begin{cases} +1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases} \quad (2.9)$$

Where  $\theta_i$  is the threshold of node  $i$ . The typical use of the Hopfield network is to error-correct a signal. i.e., given a set of signals, a Hopfield network is then trained to memorize all of the signals. A corrupted signal is then fed into the network, and the error-corrected signal is the expected output of the network.

The limitation of the Hopfield network is that it cannot keep the memory very efficient. A network of  $N$  units can only store memory up to  $0.15N^2$  bits.

### 2.1.7 Boltzmann Machine (BM)

BM [82] proposed by Ackley et al. in 1985. It is a Hopfield network with hidden units, as shown in Figure 2.5.

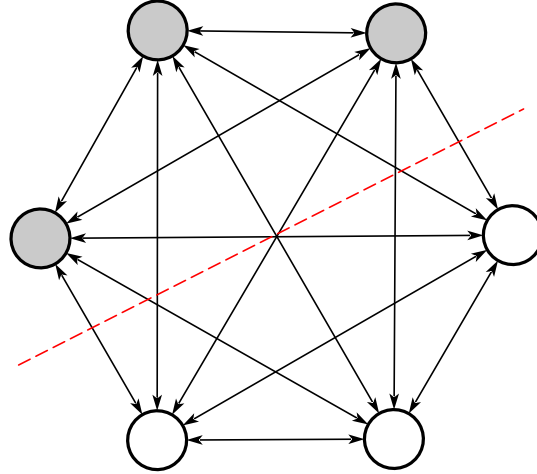
As shown in the figure, by having hidden units, BM can somewhat bypass the memory restriction Hopfield network. In BM, only the normal (i.e., visible) nodes are used for input and output purposes. Hidden nodes are merely used to increase the capacity of the network.

Besides introducing hidden nodes, BM different from the Hopfield network in the updating process. Instead of using binary step as activation function, BM utilize probability.

$$p_{i=\text{on}} = \frac{1}{1 + \exp(-\frac{\Delta E_i}{T})} \quad (2.10)$$

Where  $T$  is a scalar referred to the temperature of the system, and  $\Delta E_i$  is the difference of energies of the two states at node  $i$ .

$$\Delta E_i = E_{i=\text{off}} - E_{i=\text{on}} \quad (2.11)$$



**FIGURE 2.5** Structure of Boltzmann Machine [82]. Filled nodes (●) are normal units. Unfilled nodes (○) are hidden units. The red dashed line divides normal and hidden units.

$$= \sum_{j>i} w_{ij} s_j + \sum_{j<i} w_{ji} s_j + \theta_i \quad (2.12)$$

Where  $\theta_i$  is the bias of node  $i$  in the global energy. The name 'Boltzmann' is derived from the fact that Equation (2.10) is the result of substituting the energy at each state in Equation (2.11) with its relative probability according to Boltzmann factor [108]<sup>2</sup>.

### 2.1.8 Restricted Boltzmann Machine (RBM)

Originally known as Hormonium [83] proposed by Smolensky in 1986. Compared to the Boltzmann Machine, there is no connection between visible nodes and invisible nodes, as shown in Figure 2.6.

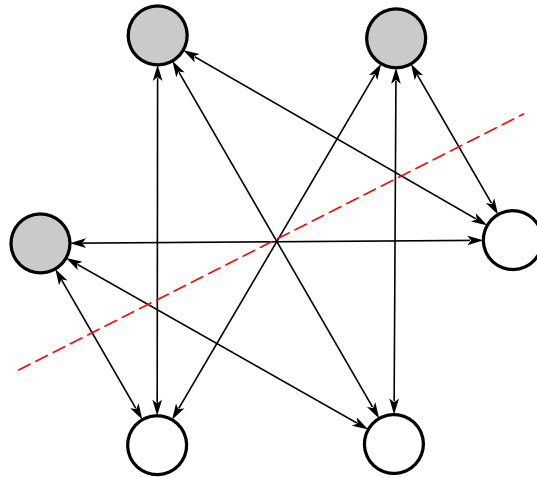
The energy of the network is as follows.

$$E(v, h) = - \sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j \quad (2.13)$$

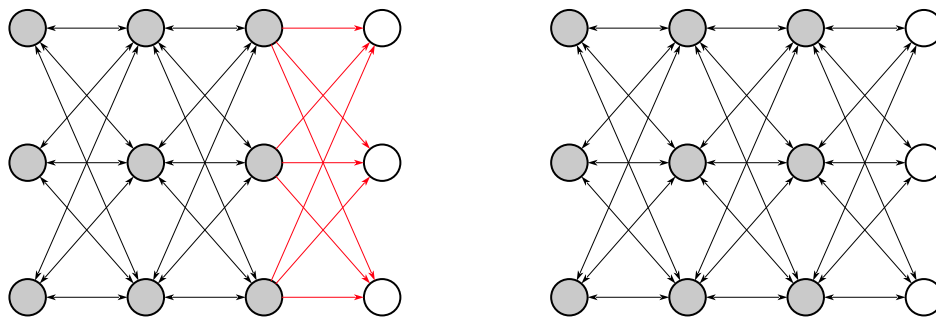
Where  $h$  is hidden node,  $v$  is visible node,  $a$  is bias for visible node,  $b$  is bias for hidden node, and  $w_{ij}$  is the weight of vertex that connect  $v_i$  and  $h_j$ . Restricted Boltzmann Machine (RBM) raised to popularity in mid-2000 with the invention of Deep Belief Nets (DBN) and fast training algorithm [109].

Figure 2.7 shows the structure of DBN [109] and Deep Boltzmann Machine (DBM) [110]. They are the two most popular variances of RBM in mid-2000. As shown in the figure, both

<sup>2</sup>Original article published in 1909 in German. The citation is the translated version published in 2015



**FIGURE 2.6** Structure of Restricted Boltzmann Machine [83]. Filled nodes (●) are normal units. Unfilled nodes (○) are hidden units. The red dashed line divides regular and hidden units. Different from BM, hidden node and visible node are not connected to each others.



**FIGURE 2.7** Deep Belief Nets (DBN) on the left and Deep Boltzmann Machine (DBM) on the right. In DBN, the connection between visible node and hidden node are not bi-directional.

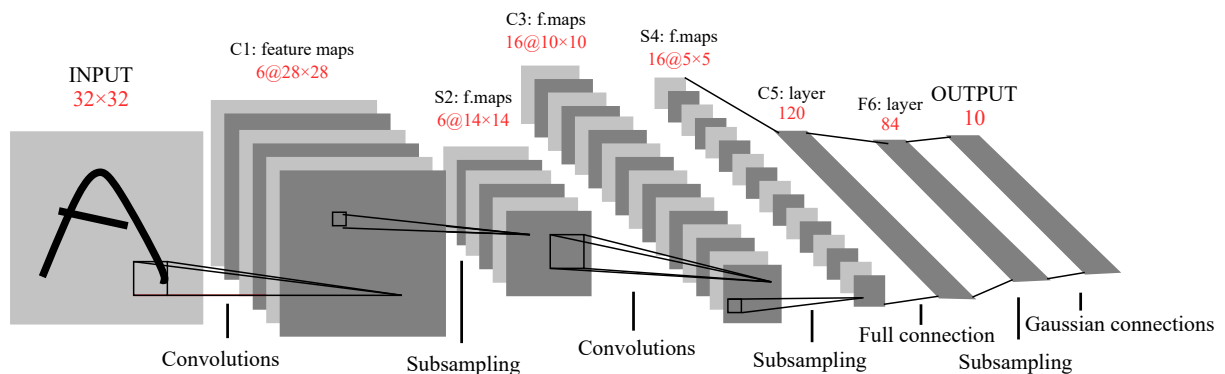
DBN and DBM are quite similar to modern DNN. DBN and DBM are similar. However, in DBN, the connections between the visible nodes and hidden nodes are one direction.

In [109], Hinton et al. showed that DBN can be stacked and trained greedily. This layer-wise training technique is still being used in popular nowadays. Moreover, the term *deep* appear to be used first in DBN. The deep autoencoder is popularized around this time [111]. Therefore, many researchers consider DBN is the beginning of the nowadays DL era.

### 2.1.9 LeNet

LeNet in general refers to LeNet5 [47, 84, 112, 113], a five layers CNN first proposed by LeCun et al. in 1989. LeNet5 is heavily inspired by Neocognitron [80], and it was one of the first CNN ever developed. As shown in [84], LeNet5 outperforms its previous works in many

aspects. It is also known for the ability to handle scale, position, rotation variation, and certain types of affine transformations. LeNet5 achieves a 0.95% error rate on MNIST [114] data set and has 60 thousand parameters.



**FIGURE 2.8** Structure of LeNet5 for character recognition [84].

The structure of LeNet5 is shown in Figure 2.8. Compared to traditional ANN, LeNet5 introduces convolutional layers and subsample layers. As shown in the figure, LeNet5 consists of two convolutional layers and two subsampling layers, followed by conventional ANN. Different from ordinary convolution operation, convolutional layers employ activation functions for non-linear transformation. Both convolutional layers in LeNet5 have a kernel size of  $5 \times 5$  pixels and utilize Sigmoid as the activation function.

The two subsampling layers gradually reduce the size of the feature maps by taking an average out of  $2 \times 2$  pixels areas. Thus, compared to modern DNN, LeNet5 is very lightweight. However, its computational resource requirement was still significant at the time. Therefore, CNN, ANN, or DNN in general are still out of favor until cheap computational resource from GPGPU available.

### 2.1.10 Other Related Works

In literature, associationism of Plato (424/423–348/347 BC) and Aristotle (384–322 BC), Neural Grouping [115] of Bain, and Self Organizing Graph [116] of Kohonen are usually considered milestones of the DL development. However, the authors find that the relationship between the above-mentioned works and DL’s line of works is not significant.

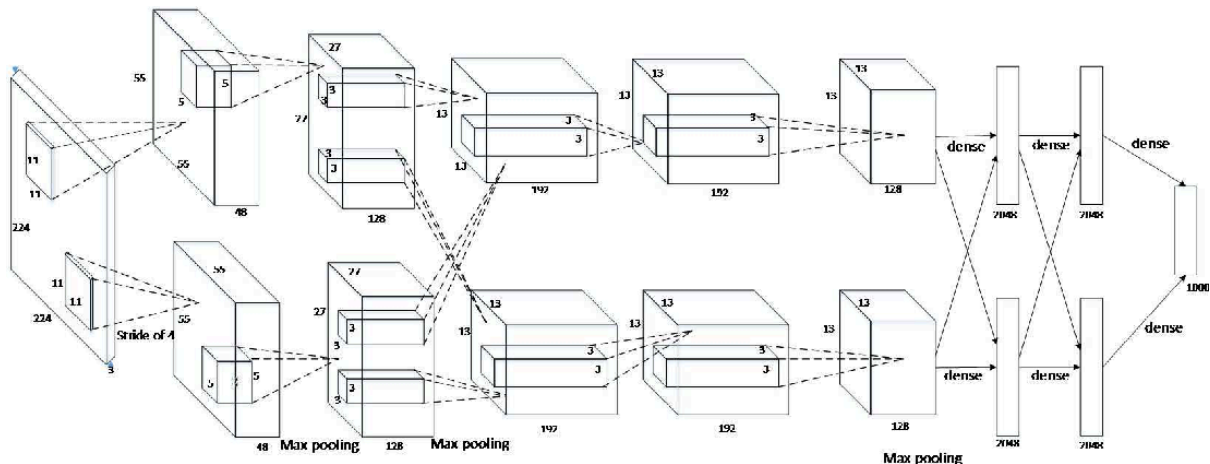
## 2.2 Deep Learning Era

After a decade long since the final paper on LeNet5 published [84], GPGPU has resurrected the interest in CNN and DNN in general. GPGPU allows DNN with many more layers to be trained. As a result, the deeper underlying feature of a given data can be generalized by the

DNN. Countless works on DL have been done in the 2010s. Nonetheless, general consensus agrees that the current DL era started with the publication of AlexNet [5].

## 2.2.1 AlexNet

Compared to LeNet5, AlexNet has eight convolutional layers with 60 million parameters. Hence, it not only has three more layers and 100 times more parameters.



**FIGURE 2.9** Structure of Original AlexNet [5]. The network consist of two branches. Each branch is trained on a GPU.

As shown in [Figure 2.9](#), AlexNet consists of two branches except for the output layer. The two branches are structurally identical with three cross-concatenation links in layers 3, 7, and 8. However, once trained, the parameters of the two branches are different. In the original paper, Krizhevsky et al. shows that the two branches learned to detect different features. At the first convolutional layer, one of them learned to detect edges. Another learned to detect color features. This design was driven by the technical limitations at the time rather than any model performance advantage. AlexNet was originally trained on two GTX 580 graphic processing units (GPUs). Each of these GPUs was equipped with 3GB of memory.

In the newer implementation of AlexNet on later generations of GPUs, the two branches are replaced by only one. The single branch design technically different from the original AlexNet design and consumes more memory. Nonetheless, evaluations show that the performance of the single branch implementation is compatible with the original implementation. [Figure 2.10](#) illustrates the new single-branch AlexNet design.

Come with AlexNet is the re-introduction of CNN and pooling layers. Compared to LeNet5, AlexNet or modern DNN in general stills heavily relies on these layers, especially in the field of CV and digital signal processing (DSP).

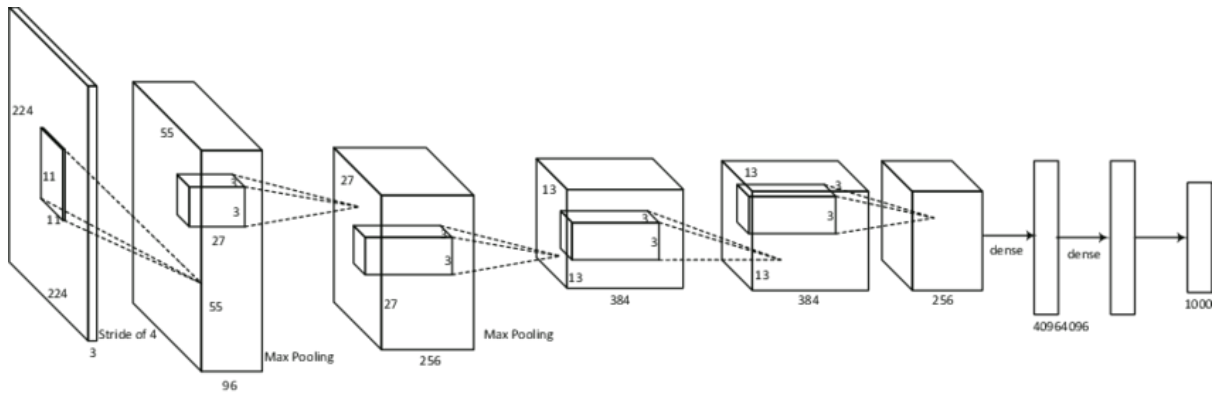


FIGURE 2.10 Structure of Single Branch AlexNet [5].

**Convolutional layer** is a crucial part of modern DNN. Figure 2.11 illustrates a convolutional operation. This convolution operation essentially is a filtering process. Thus, it responds to the pattern that similar to the filter itself. A convolutional layer usually made up of multiple filters like this. Different from the figure, a filter in practice also operates in a multi-channel input.

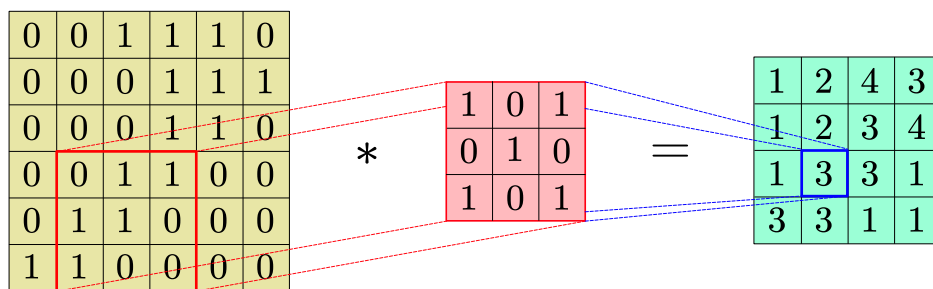


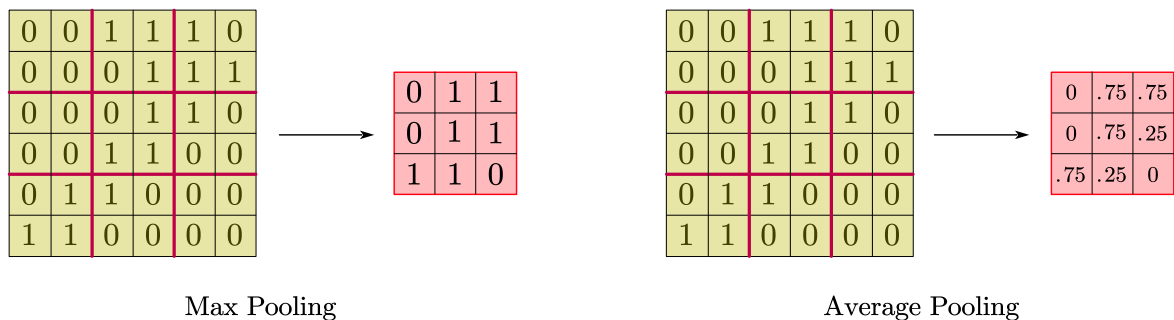
FIGURE 2.11 Visualization of a Convolution Operation.

**Pooling layer** is another essential ANN layer. Figure 2.12 illustrates the max and average pooling operations. Similar to the convolutional layer, pooling also has a kernel. However, instead of performing dot products, pooling kernels perform either max or average computation, which is computationally inexpensive. In practice, max pooling is more popular and mostly used to reduce the size of the input.

## 2.2.2 VGG Neural Networks

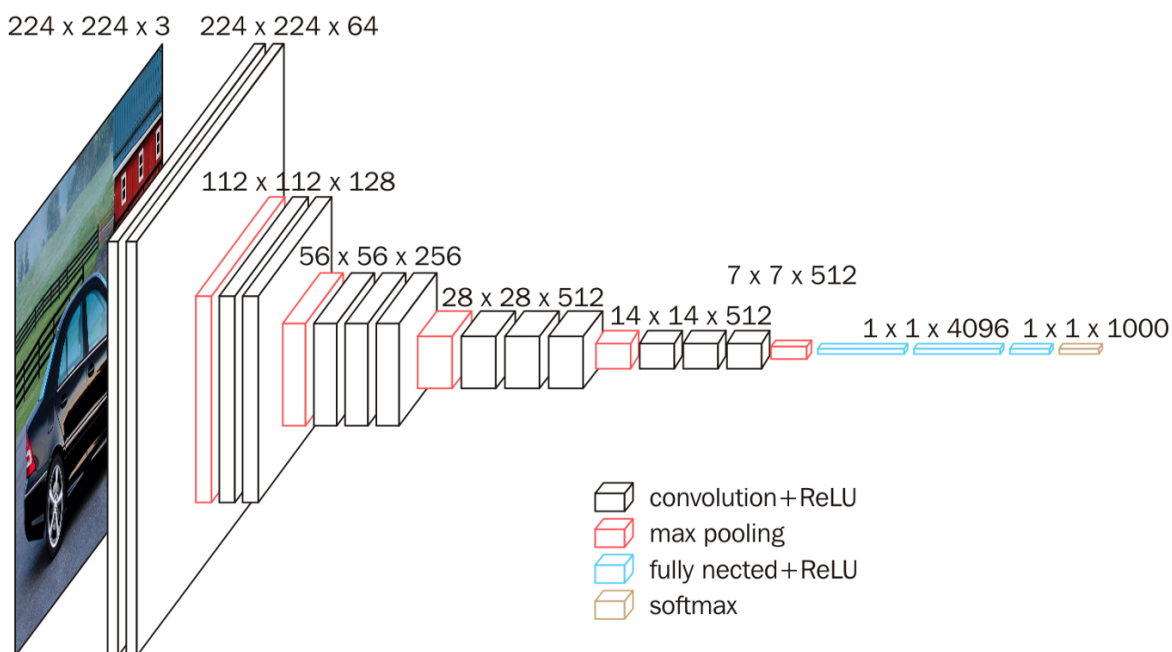
Visual Geometry Group (VGG) is a special interest group of Oxford University, United Kingdom. Simonyan and Zisserman proposed the VGG net [63] in 2015. VGG net achieves state of the art performance by increasing the number of convolutional layers up to 19. VGG





**FIGURE 2.12** Visualization of Pooling Operations.

net is characterized by its simplicity in design. The network uses only  $3 \times 3$  pixel filter size for CNN layers. Subsampling is done by only max pooling.



**FIGURE 2.13** Structure of VGG16 [63].

In [63], the authors proposed two designs: VGG16 and VGG19. VGG16 and VGG19 have 16 and 19 layers correspondingly. Figure 2.13 shows the structure of VGG16. To train VGG, the authors employed a technique named pre-training. Training the full-length network would require a shorter version of it to be trained. The weight of the short trained network is then used to initialize the longer version of the network.

Therefore, the main criticisms for VGG nets also include training time. At the time of its publication, VGG is considered a large network. The pre-trained weights of VGG16 and VGG19 are approximately 530MB and 570MB. However, it still owns a fair share of popularity

due to the simplicity in design. Pre-trained VGG nets are frequently used to extract image features as in 2020.

### 2.2.3 Inception Net

Inception Net first introduced in 2015 by Szegedy et al. as GoogLeNet. Inception-v1 (i.e., GoogLeNet) was the winner of the ILSVRC challenge in 2014. Inception-v1 comprise of multiple inception modules, in which different size filters are used to extract features at the same time. Figure 2.14 shows the structure of inception module.

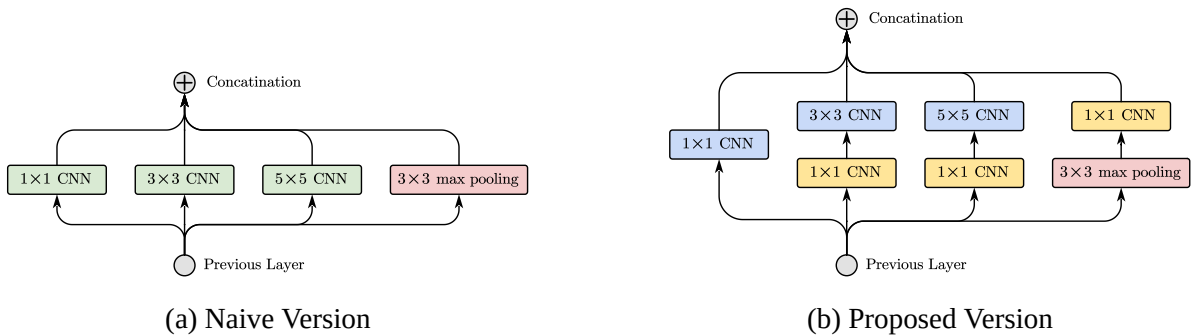
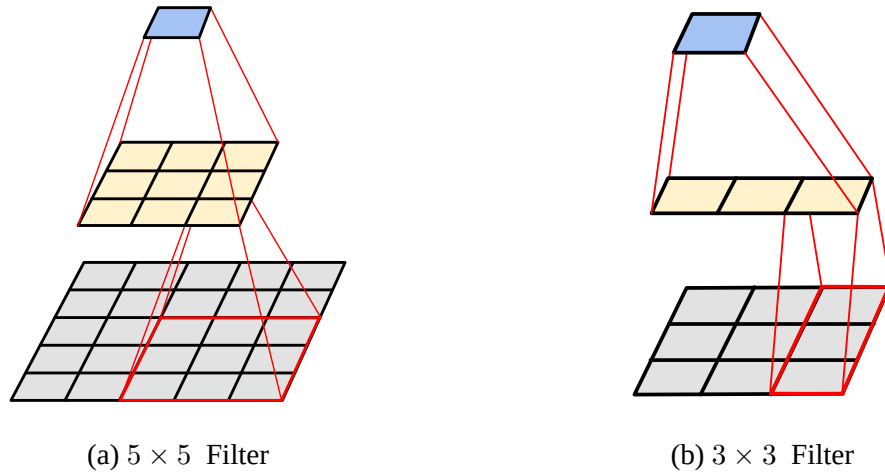


FIGURE 2.14 Inception Modules [13]

As shown in Figure 2.14a, the outputs of all the branches of filter are concatenated. Hence, the output of the module is unrealistically large for the next module to process. To overcome the difficulty,  $1 \times 1$  CNN blocks are used to reduce the dimension of the feature. The better version of this inception module is shown in Figure 2.14b.

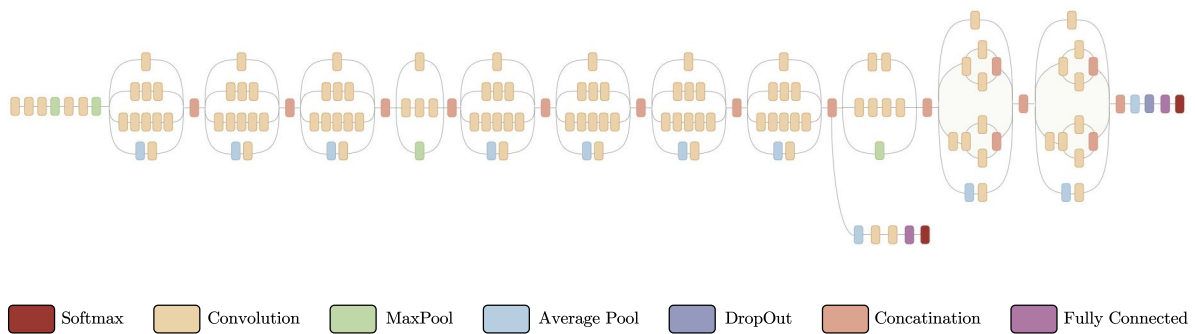
Inception-v2 [14] was published in 2015 by Ioffe and Szegedy. However, the main focus of this paper was batch normalization (BN). Therefore, inception-v2 is also known as BN-Inception. BN improves the training speed as well as the stability of the training process. BN normalizes the input of a layer by ensuring the mean and variance of each batch of input is fixed. The reason for the effectiveness of BN is the mitigation of internal covariant shifts [14]. Thus, eliminate the effect of vanishing or exploding gradient [117, 118]. However, a more recent study shows that BN rather smoothen objective function [119]. Another recent study claimed that batch normalization achieves length-direction decoupling instead [120]. Overall, inception-v2 was able to reduce the top-1 error rate by nearly 4% on ImageNet challenge.

Inception-v3 [15] was published in 2016 by Szegedy et al.. Inception-v3 introduces factorization convolutions. The purpose of factorization convolution is to reduce the number of parameters without reducing the efficiency of the network. Shown in Figure 2.14, inception net uses  $3 \times 3$  and  $5 \times 5$  CNN layers. Both of the two filter are replaced by a factorized version, as shown in Figure 2.15.  $5 \times 5$  CNN filters are replaced by two  $3 \times 3$  filters as in Figure 2.15a.



**FIGURE 2.15** Factorization Convolution [15]

Similarly,  $3 \times 3$  filters are replaced by  $3 \times 1$  and  $1 \times 3$  filters. In such a way, the number of connections is reduced by approximately 28%.



**FIGURE 2.16** Structure of Inception-v3 [15].

Figure 2.16 shows the structure of Inception-v3. Can be seen on the figure,  $3 \times 3$  filters have been replaced by two convolutional filters. As mentioned,  $5 \times 5$  filters can be replaced by two  $3 \times 3$ . In turn, each  $3 \times 3$  filter can be replaced by another two filters. Thus, each  $5 \times 5$  filters can be replaced by four other filters. Hence, the number of connections can be reduced by 35.85%.

Inception-v4 [16] in released in 2017 further improves the performance of the network. Compared to inception-v3, inception-v4 reduce the top-4 error rate by 0.48%. This was done by implementing skip connection (will be mentioned in Section 2.2.4) and architecture refinement.

## 2.2.4 Residual Neural Network (ResNet)

First introduced in 2016 by He et al., Residual Neural Network (ResNet) [64, 121] is a 152 layers network. Hence, it is about ten times deeper than any other network ever introduced at

the time.

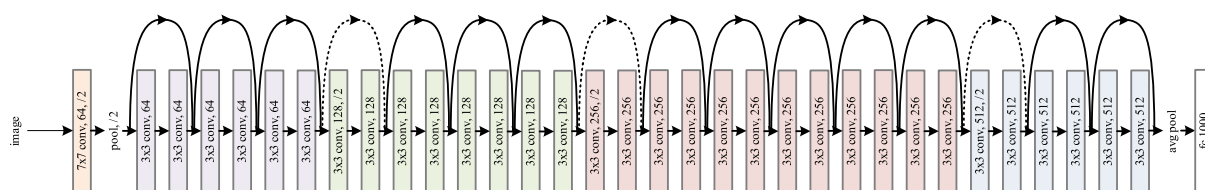


FIGURE 2.17 Structure of ResNet [64].

Figure 2.17 shows the structure of a 34 layers ResNet. Popular configurations of ResNet are 34, 50, 101 and 152 layers. The deeper versions of ResNet share the same design with ResNet-34, with the only difference is the number of layers. ResNet adopts simple filter designs that similar to VGG.

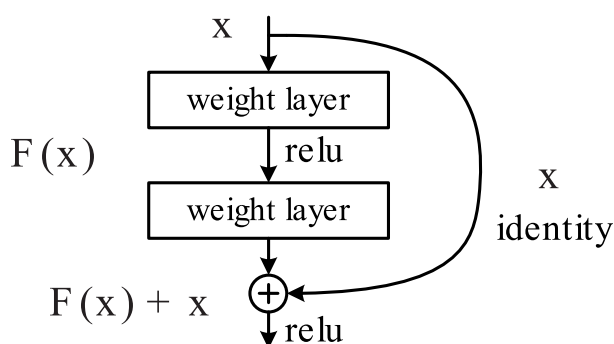


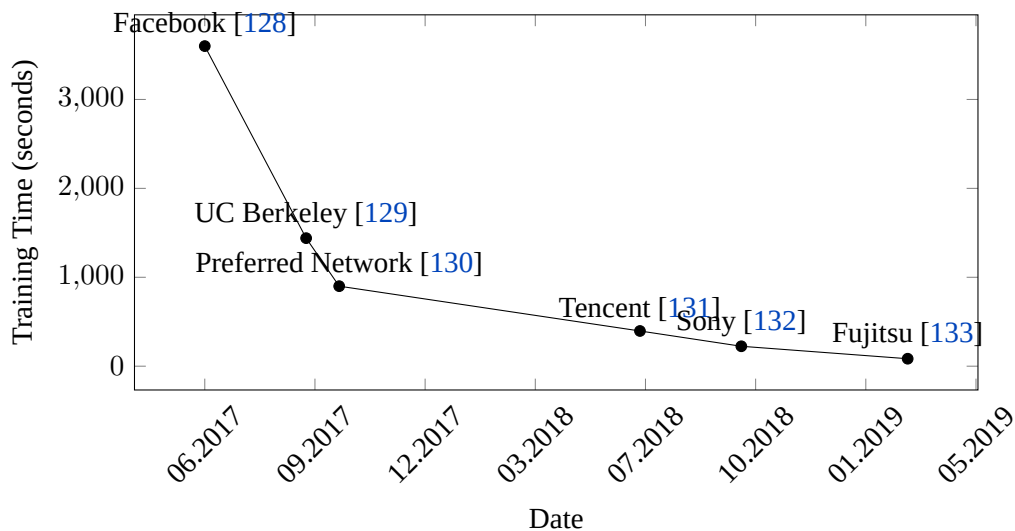
FIGURE 2.18 Structure of A Residual Block [64].

As shown by the work on VGG [122], deep networks are difficult to train. To overcome this difficulty, ResNet introduces Residual Block. Residual Block essentially is a block of several CNN layers with an identity connection passing the input to the output. Figure 2.18 shows the structure of a Residual Block.

Identity connection, as known as skip connection, is popular in more recent models. Similar to real brain structures [123–125], skip connections pass outputs directly from a layer to distance layers by skipping their intermediate layers. Because it mitigates the negative effect of vanishing gradient, offers more comprehensive input for deeper layers, and speeds up the training process. Most recent success, such as HighwayNets [122, 126], Densely Connected CNN [127], or U-Net [65] are all utilizing skip connections.

## 2.2.5 Multi-GPU Training

With the success of DL, new data sets tend to be larger and higher in quality. Moreover, DNN models are also getting more sophisticated. Therefore, it is commons to train DNN in distributed settings.



**FIGURE 2.19** The Distributed Training Race: in just one and a half years, the time required to train ResNet50 [64] has fallen from one hour down to only 84 seconds with distributed training.

Distributed training in a broad sense also includes the multi-nodes (machines) setting. For business, having a model trained in a short time is critical because it needs to be updated to fit new data better. Therefore, distributed training received much more attention from private businesses and cloud service providers. Around the time the ImageNet challenge discontinued in 2017, there was an unofficial competition for shorter DNN training time.

In June 2017, Goyal et al. from Facebook reports they have trained ResNet50 in one hour using 256 Nvidia Tesla P100 GPU [128]. In September of the same year, You et al. from UC Berkeley university reported 24 minutes training time using 2048 Intel Knight Landing chipsets [129]. Two months later, Akiba et al. from Preferred Network reported 15 minutes training time using 1024 Nvidia Tesla P100 GPU [130]. In July 2018, Jia et al. from Tencent reported 6 minutes 36 seconds training time using 2048 Nvidia Tesla P40 GPU [131]. Beside distributed training, the authors also employ new technical called mixed-precision training to reduce the computation cost (see Section 2.4.1). In November 2018, Mikami et al. from Sony reported 3 minutes and 44 seconds training time with 2176 Nvidia Tesla V100 GPU [132]. And finally, in March 2019, Yamazaki et al. from Fujitsu reported only 84 seconds training time using 2048 Nvidia Tesla V100 GPU [133]. Thus, in just one and a half years, the time required to train the ResNet50 on ImageNet has reduced to 2.33%.

On the one hand, this technical achievement is impressive. On the other hand, such a level of optimization requires extra effort. Therefore, it is only meant for the business's deployment. The dominant setting for the research activity is single-node, multi-GPUs. There are two main strategies for parallel training:

**Model parallelism** is the strategy where the model is designed to works on multiple GPUs. A

classic example of this strategy is AlexNet [5].

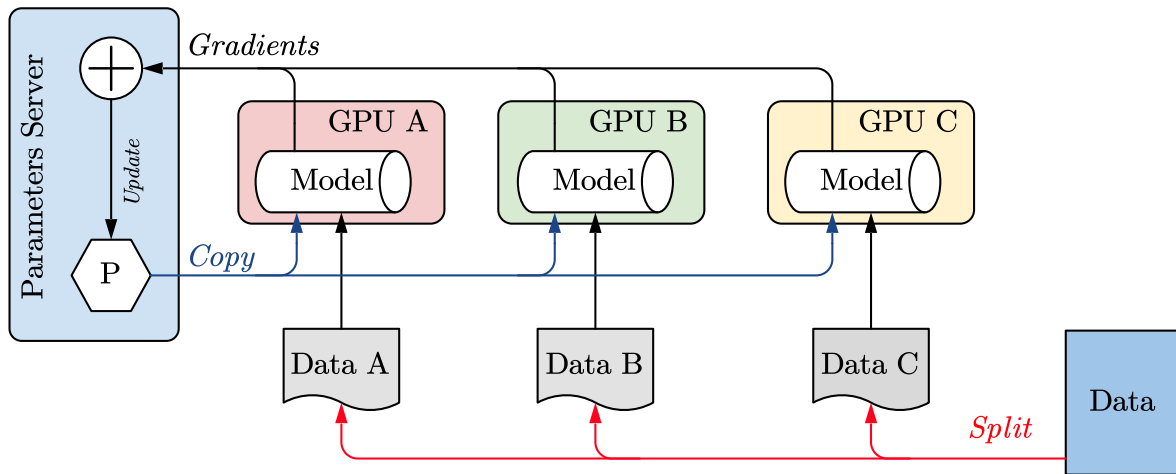


FIGURE 2.20 Data Parallelism Multi-GPU Training.

**Data parallelism** In this strategy, the model is cloned to each GPU, and input data is split into each GPU. As shown in Figure 2.20, each GPU receives a chunk of input each iteration. After calculating, the gradients from all the GPUs are sent to the parameter device for updating. The new parameter values are then cloned to all the GPUs before the new training iteration.

---

**Algorithm 1** Multi-GPU Training with Data Parallelism

---

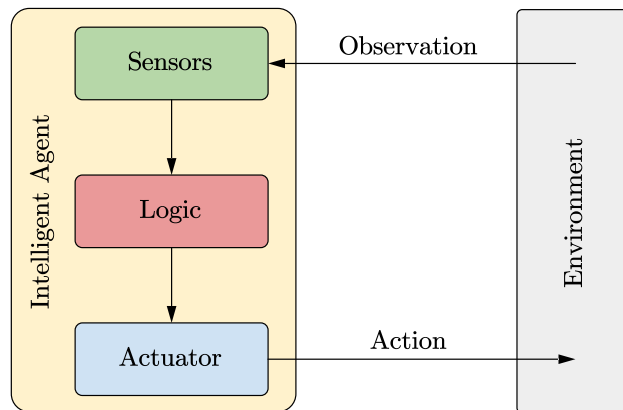
- 1: **given** Learning Rate  $\alpha$
  - 2: **initialize**  $\mathbf{x}_1 \leftarrow \text{Random}()$  ▷ Initialize  $\mathbf{x}_1$  on GPU-1
  - 3: **repeat**
  - 4:      $\mathbf{x}_2 \leftarrow \mathbf{x}_1$  ▷ Copy parameter from GPU-1 to GPU-2
  - 5:      $\mathbf{x}_3 \leftarrow \mathbf{x}_1$  ▷ Copy parameter from GPU-1 to GPU-3
  - 6:      $b \leftarrow \text{GetNewTrainingBatch}()$  ▷ Get new training mini-batch
  - 7:      $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \leftarrow \text{Split}(b)$  ▷ Split the mini-batch and send to each GPU
  - 8:      $\Delta \mathbf{x}_1 = \text{BackProp}(\mathbf{x}_1(\mathbf{b}_1))$  ▷ Compute gradient of batch 1 on GPU-1
  - 9:      $\Delta \mathbf{x}_2 = \text{BackProp}(\mathbf{x}_2(\mathbf{b}_2))$  ▷ Compute gradient of batch 2 on GPU-2
  - 10:      $\Delta \mathbf{x}_3 = \text{BackProp}(\mathbf{x}_3(\mathbf{b}_3))$  ▷ Compute gradient of batch 3 on GPU-3
  - 11:      $\Delta \mathbf{x}_1 = \text{ReduceAverage}(\Delta \mathbf{x}_1, \Delta \mathbf{x}_2, \Delta \mathbf{x}_3)$  ▷ Average the gradient to GPU-1
  - 12:      $\mathbf{x}_1 \leftarrow \mathbf{x}_1 + \alpha \Delta \mathbf{x}_1$  ▷ Update parameters on GPU-1
  - 13: **until** stopping criterion is met
  - 14: **return** optimized parameters  $\mathbf{x}_1$
- 

In some newer generations of consumer-grade GPUs, one of the GPUs can act as a parameter device. With GPU make by Nvidia, NVIDIA Collective Communications Library (NCCL) allows efficient cross-device reduction operations through NV-Link. Thus, it

allows for more accessible multi-GPU training compared to the previous generation of GPUs. [Algorithm 1](#) shows the multi-GPU training routine with data parallelism.

## 2.3 Intelligent Agent

An intelligent agent (IA) is an autonomous entity within an environment. IA acts, directing towards achieving goals. [Figure 2.21](#) shows the structure of a general IA and its interaction with the environment. In practice, the agent can be as simple as condition-action rules. Russell and Norvig classify agents into five classes [\[134\]](#): **a)** simple reflex agents **b)** model-based reflex agents **c)** goal-based agents **d)** utility-based agents **e)** learning agents. Thus, the interaction between the agent and the environment over time toward a goal is the main difference between an IA and a plain deep ANN. We find this scenario is useful, especially when an end-to-end solution is not possible.



**FIGURE 2.21** A General intelligent agent.

The goal that needs to be achieved by the IA is typically an NP-hard problem. Therefore, there is no ground truth for training except the observations from the environment over time. This leads to the use of reinforcement learning (RL). Instead of fitting a model to a set of given ground-truth, RL improves the performance of the IA through numerous trials. In the context of RL, a trial is called an episode. Each episode comprises multiple steps along the timeline. At every step, the IA observe the environment and perform an action.

With the rise of DNN, the field of RL also achieved notable success recently. Mnih et al. introduced deep Q-network (DQN) [\[27\]](#) that plays Atari 2600 games well above the skill of a human player or any ever proposed linear models. Subsequently, the works on prioritized experience replay [\[135\]](#), double Q-network [\[136\]](#), duel Q-network [\[137\]](#), and asynchronous actor-critic method [\[38\]](#) further enhance the efficiency of the training process. AlphaGo [\[39\]](#), an IA developed by Silver et al. defeated the world best human player in Go [\[138\]](#). Lample

and Chaplot proposed an IA that achieved human-level in playing Doom [140], a first person shooter (FPS) game [139].

There are two major categories of RL algorithms. Algorithms in the first category decide an action to act based on the estimated scores given for each action. Algorithms in the second category estimate the probabilities that it should take on each action. This probability is called advantage. Mathematically, given action  $i$ , the value-based algorithms estimate the score  $Q_i$ . The probability action  $i$  will be performed is  $p_i = f(Q_i)$ . Where  $f(\cdot)$  is the function to convert a score to the probability. This function is hand-designed. On the other hand, the advantage based algorithms estimate  $p_i$  directly.

### 2.3.1 Q-Learning

In the typical setting of Q-learning, a reward  $r_t$  is given to the IA after every time step  $t$ . Because different actions lead to different consequences, an accurate metric to measure the performance of the IA would be the sum of discounted rewards (SDR). Given the episode length is  $T$ , the SDR is calculated as follows:

$$R_t = \sum_{i=t}^T \gamma^{i-t} r_i \quad (2.14)$$

Where  $R_t$  is the SDR at time step  $t$  and  $0 \leq \gamma \leq 1$  is the discount factor. The higher this value, the more important the future reward. Thus, tweaking  $\gamma$  would affect how the IA prioritize each action for long or short term gains. Since  $R_t$  can't be given to the IA during inference, it has to estimate this value as follows:

$$Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a] \quad (2.15)$$

Where  $Q^\pi(s, a)$  is the estimated value of  $R_t$  given the observation (i.e. state)  $s$  and the action performed is  $a$  according to policy  $\pi$ . The optimal policy  $\bar{\pi}$  is achieved when the estimated  $Q$  value for each action is maximized.

$$Q^{\bar{\pi}}(s, a) = \max_{\pi} Q^\pi(s, a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a] \quad (2.16)$$

With (2.14) and (2.15), (2.16) can be written as follows:

$$Q^{\bar{\pi}}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} [Q^{\bar{\pi}}(s', a')] \mid s, a \right] \quad (2.17)$$

Where  $s'$  and  $a'$  are the state and the action of subsequent time step, respectively, and  $r$  is the immediate reward for action  $a$  given in state  $s$ . In deep Q-learning, this value is approximated by an DNN parameterized by  $\theta$ .



$$Q^\theta(s, a) \approx Q^{\bar{\pi}}(s, a) \quad (2.18)$$

Assume that the model  $\theta$  is trained, at time step  $t$ , the  $Q$  value of the action  $a_t$  must be close to the sum of the immediate reward  $r_t$  and the discounted maximum of the SDRs of all the possible actions in the subsequent time step.

$$Q^\theta(s_t, a_t) \approx r_t + \gamma \max_a [Q^\theta(s_{t+1}, a)] \quad (2.19)$$

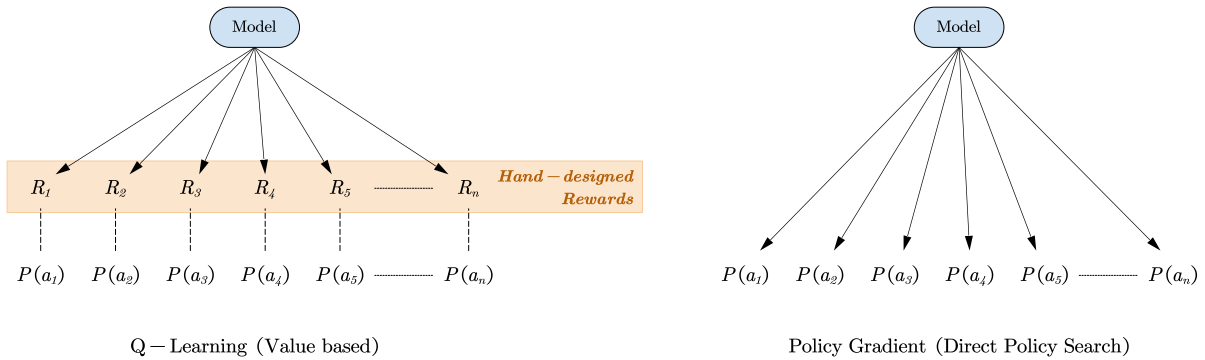
Given that  $y_t = r_t + \gamma \max_a [Q^\theta(s_{t+1}, a)]$ , the loss function is defined as follows:

$$L_t = |y_t - Q^\theta(s_t, a_t)|^2 \quad (2.20)$$

Where  $L_t$  is the loss value at time step  $t$ . Notice that  $y_t$  is the right hand side of equation (2.19). Thus, (2.19) can be rewritten as  $Q^\theta(s_t, a_t) = \delta + y_t$  where  $\delta$  is the approximation error. In consequence, equation (2.20) can be written as  $L_t = |\delta|^2$ .

Hence, training the model is actually making the model improve the  $Q$  value estimation without contradicting itself. For example, action  $a_t$  at time step  $t$  results in the change of the environment from  $s_t$  to  $s_{t+1}$  and reward  $r_t$ . Given a model at any stage, all the components in equation (2.19) are already given or can be obtained effortlessly. Therefore, the model can be trained to minimize the difference between the two sides of the equation.

### 2.3.2 Policy Gradient



**FIGURE 2.22** Comparison Between Q-Learning and Policy Gradient.

Figure 2.22 shows the comparison between Q-Learning and policy gradient learning. IAs trained by the Q-learning method predicts the state-action values of all the action in the action space. Then action is chosen deterministically based on the highest value. Thus, it heavily depends on the value function in every step to result in better policy approximation. On the con-

trary, with the policy-gradient method, the IAs is trained to output the action directly. Formally, policy-gradient optimizes policy  $\theta$  to maximize the expected SDR  $R_t$ :

$$\theta = \arg \max_{\theta} \mathbb{E} [ R_t ] \quad (2.21)$$

To optimize the policy  $\theta$ , the gradient of policy is given by:

$$\nabla_{\theta} \mathbb{E} [ R_t ] = \mathbb{E} [ \nabla_{\theta} \log P(a_t) R_t ] \quad (2.22)$$

Where  $P(a_t)$  is the probability of action  $a_t$  at time step  $t$ . Thus, actions that lead to better SDR  $R_t$  are encouraged. In order to train an agent using policy-gradient,  $R_t$  must be known or has to be approximated.

### 2.3.3 Intelligent Agent (IA) Training Loop

A naive IA training strategy is alternatively letting the IA interacts with the environment for one step and train the IA with the collected experience. It is easy for the IA to be biased to a small set of actions. Thus, trapping itself into a sub-optimal solution. Therefore, this naive strategy would mostly lead to failure.

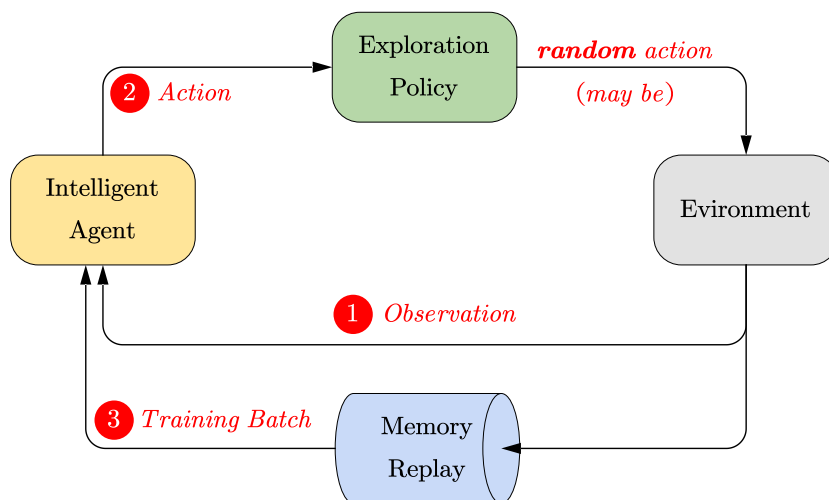


FIGURE 2.23 The IA Training Loop

Figure 2.23 shows a typical IA training loop. Compared to the standard setting, there are two additional components: memory replay and exploration policy. There are three major steps in this loop. Step 1 and 2 are similar to the above-mentioned naive strategy. However, in step 1, the observation is sent to memory replay and IA at the same time. The IA is periodically trained after repeating steps 1 and 2 for a predefined number of times.

### 2.3.4 Experience Memory Replay

One of the most significant difficulties in training an IA is the strong correlation between the network policy and the action-outcome of subsequent time steps. This difficulty makes online training impossible. To break the strong correlation, experience memory replay (EMR) [135] is used.

Regardless of the algorithms used, data is crucial for the training process. To bootstrap this data, the IA is used to unroll a certain amount of episodes. The result of the episode as a whole is trivial since it is not required for training. Thus, at every time step  $t$ , the experience  $(s_t, a_t, r_t, s_{t+1})$  of the IA is stored in the replay memory. This memory is a queue with a large capacity. In popular works like DQN [27], this memory is set to store  $1 \times 10^6$  samples of such experience. Because this memory is a queue, old experiences are discarded to accommodate new ones when it is full. Mini batches of random experiences are used to train the IA.

### 2.3.5 Exploration and Exploitation

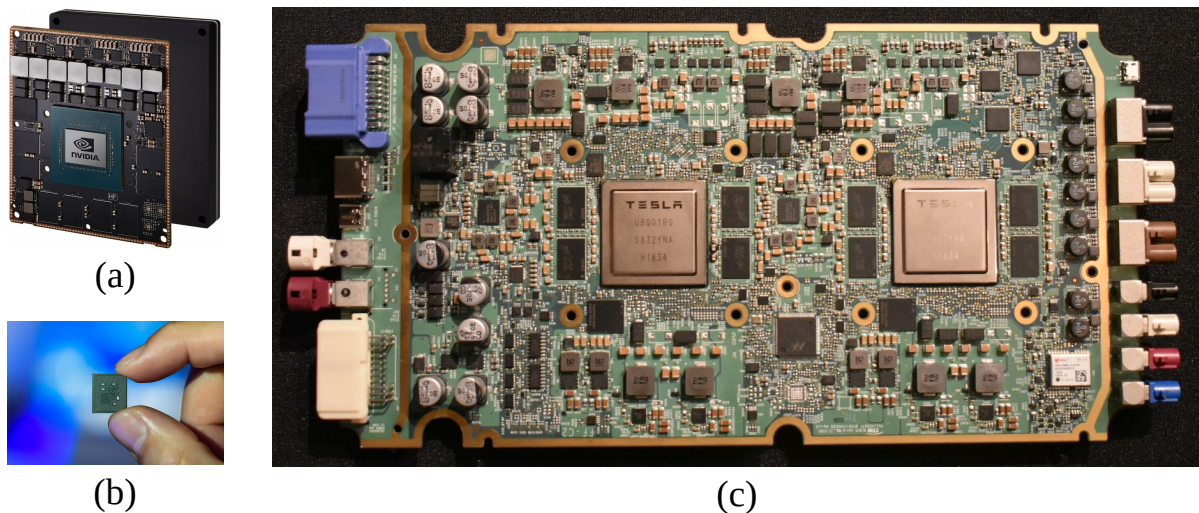
Training an IA in RL requires a right balance between exploitation and exploration. Exploitation is relying on the learned policy to improve the prediction accuracy while exploration allows the IA to seek for potentially better solutions (i.e., avoiding sub-optimal trap). A popular exploration policy being used in RL is  $\epsilon$ -greedy [141]. Under this policy, the output of the IA has an  $\epsilon$  chance of being random.

There are other exploration policies based on randomization, such as Thompson sampling [142] and Bayesian sampling [143]. However, a variance of  $\epsilon$ -greedy policy named reducing  $\epsilon$ -greedy is commonly used. With this policy, the IA starts with high exploration rate, and then gradually reduces it towards the end of the training process. Thus, it allows the agent to explore more (and avoiding extreme bias) in the beginning, and to focus more on exploitation in the later phase of the training process.

## 2.4 Model-Design v.s. End-to-end network

The achievements obtained in the decade are truly remarkable. In the field of CV, the objective was constantly elevated. DL started with feed-forward networks with simple objectives like image classification. By the end of the decade, researchers are working on complex problems such as semantic segmentation or real-time image generation.

Toward the end of the decades, the end-to-end model design is highly desirable because it reduces the complexity of the model as a whole [144]. Furthermore, it can avoid sub-optimal results because of hand-designed components. In end-to-end design, the model takes input and



**FIGURE 2.24** SoCs with Accelerated Computation Capability. **(a)** Nvidia Jetson AGX Xavier module. As in 2020, the newest version of this module has power rating at 10W. It delivers up to 32 TOPs in performance. **(b)** Qualcomm Snapdragon 860 SoC released in early 2020. It can deliver up to 15 TOPs in performance. The whole SoC itself has power rating at 7W. **(c)** Tesla’s FSD computer board manufactured by Samsung. The board’s TDP is 36W. It deliver 600 GFLOPS and 74 TOPs in performance.

Image (a) and (c) courtesy of Nvidia. Image (b) courtesy of Qualcomm.

gives the output as the final result. As opposed to the end-to-end model, traditional models take the output of a DNN as one of its inputs.

For example, in the traditional model-design self-driving system, deep convolutional neural network (DCNN) can be used to recognize the surrounding objects. The model of the surrounding environment is then be built using the recognition output. The final decisions to the steering wheel of the vehicle are then issued based on the model of the environment. On the other hand, recent researches on the end-to-end self-driving model show more success [28]. End-to-end networks take sensory input as it is and output commands to the steering wheel directly. Thus, simplify the model structure and achieve better training results via back-propagation.

Nonetheless, end-to-end is not possible (or very difficult) in numbers of applications.

### 2.4.1 The Edge-Computing Trend

Cloud computing has been a trend since the late 2000s. In the CV community, it is commons to see the use of cloud services for model training. For business, cloud computing allows state-of-the-art CV products to be delivered to the end customer.

Nevertheless, because of the increase in raw data size, there is a growing interest (again) in the edge computing concept [145–150]. By definition, edge computing brings computation and data storage closer to the location where it is needed to improve latency and optimize bandwidth. For example, it is more relevant to have a CV computational unit locally to process data from

a multi-camera surveillance system. Transmitting data from such systems to the cloud is just technically inefficient. In more critical systems like self-driving, it is not feasible to utilize cloud computing to provide low latency response.

On the technical end, new SoC from leading manufactures like Apple, Qualcomm started to embed specialized processor cores for accelerated computation. [Figure 2.24](#) shows various SoCs with accelerated computational capability. As shown in the figure, the smallest SoC is Snapdragon 865. It consumes 7W of power and can deliver 15 TOPs. For comparison, LeNet5 [84] was published in 1998. Around that time, the fastest supercomputer in the world delivered around 1000 GFLOPS in performance. Hence, the Snapdragon 860 as in **(b)** is relatively faster than the top supercomputer at that time.<sup>3</sup>

Conventionally, this type of processor works only with 32-bit numbers. These processors are capable of working with the short-bit number by zeros padding the input number. This leads to negative gains in performance as a whole. The newer generation of GPUs and DSP units have become more efficient. For example, Nvidia's Turing architecture offers a linear gain in working with short-bit numbers. Thus, working with 16 bits number is about two times faster than working with 32 bits number and so on.

It leads to the growing interest in mixed precision DNN [131, 151–154]. In the mixed-precision model, most parts of the model are working with small numbers while the rest are still working with 32 bits number. It results in significant gains in performance with minimal loss in model accuracy. Mixed precision models are popular on mobile devices.

---

<sup>3</sup>The new processors are rated in TOPs. It doesn't mean that they can only compute the integer number. Most of the new processors support multiple types of numbers.

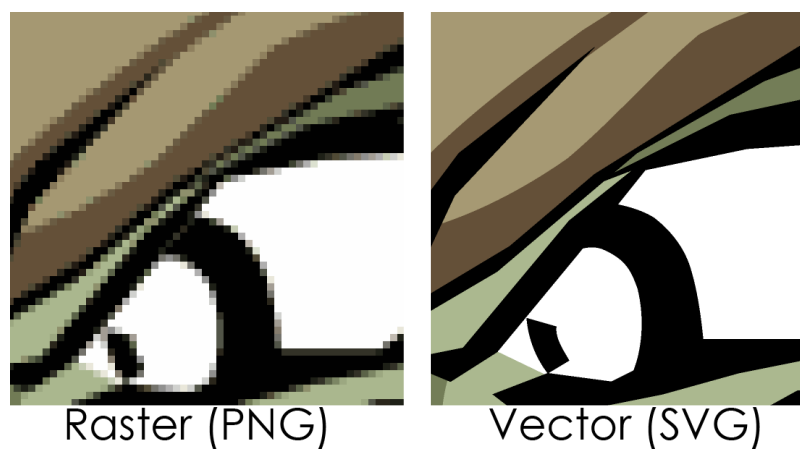


## CHAPTER 3

# SvgAI<sup>1</sup>

### 3.1 Vector Image

Figure 3.1 shows the comparison between raster and vector versions of the same image. As shown in the figure, the raster version is comprised of individual pixels. On the other hand, the vector version is crisper with all the detail preserved. A higher resolution of the raster version can be extracted. However, the size of the photo would inflate rapidly. Thus, the vector image is simply more versatile and compact. Therefore, even though a raster image is straight forward and easy to edit, raster to vector (R2V) conversion is a well-investigated topic.

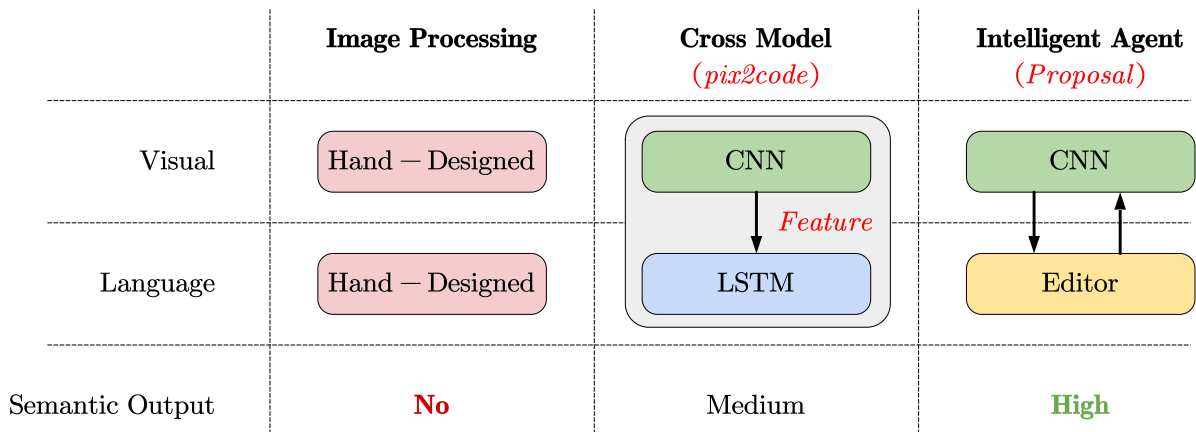


**FIGURE 3.1** Raster and Vector Images Comparison.

Image courtesy of Wikimedia Commons.

There are countless number of format for both raster and vector images. Popular format for raster images includes Joint Photographic Experts Group (JPG), Windows bitmap (BMP), Portable Network Graphics (PNG). However, for raster image, Scalable Vector Graphics (SVG) appears to be the only popular format since others such as Adobe Illustrator Artwork (\*.ai), CorelDRAW (CDR), Vector Markup Language (VML) are either proprietary or too unpopular.

<sup>1</sup>This chapter refers to the author's publication [8].



**FIGURE 3.2** Comparison Between SvgAI and Previous Works

For that reason, SVG is an synonym to vector image most of the time. In this research, we use the SVG format.

Not storing pixel by pixel information like raster images, SVG is an Extensible Markup Language (XML) based text document. The SVG standard defines numbers of elements divided into 14 functional feature sets, including paths, basic shapes, text, colors. Thus, the structure of an SVG is highly semantic. Creating an optimal vector image requires a high degree of semantic image understanding.

### 3.2 Previous Works

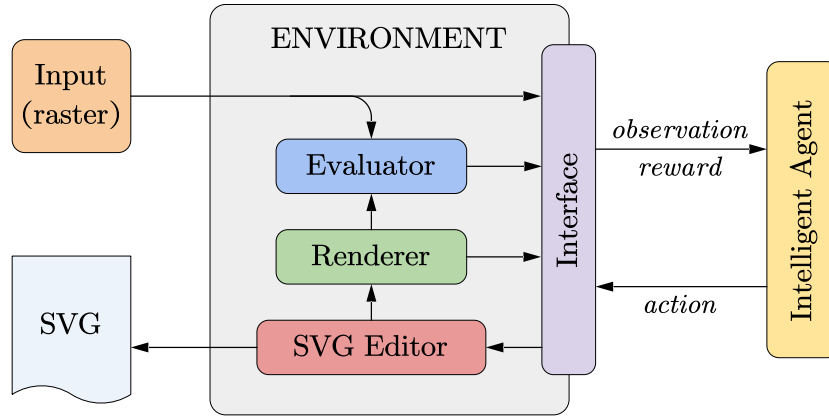
As mentioned in [Section 1.2.2](#), the related works includes conventional image processing method and a convolutional neural network (CNN)-Long-Short Term Memory (LSTM) cross model method [9]. However, both of the previous works are not producing semantic output.

[Figure 3.2](#) compares between Scalable Vector Graphic AI (SvgAI) the the previous works. The conventional image processing methods do not handle visual and SVG language at the semantic level. As a result, it does not produce semantic SVG output. [9] handles visual and language model using cross modal framework as seen in [10]. However, the accuracy and semantic property of this model is limited because LSTM does not handle hierarchical structure language well. Compared to the previous works, SvgAI does not handle the language model. Instead of generating SVG document, it sends commands to an SVG editor to draw the desired image. Thus, it simplified the model design and allowed the model to focus only on visual structure.



### 3.3 SVG Editor Environment

We propose a framework to train an intelligent agent (IA) to use SVG editor (hereafter referred as editor) with reinforcement learning (RL). The objective of this IA is to draw an SVG image that is similar to as much as possible to a given target raster image. It can be considered a new paradigm to solve the R2V problem. A custom editor is created for carrying out the research, which has modeled after OpenAI Gym [155] environment due to its robustness in interface design.

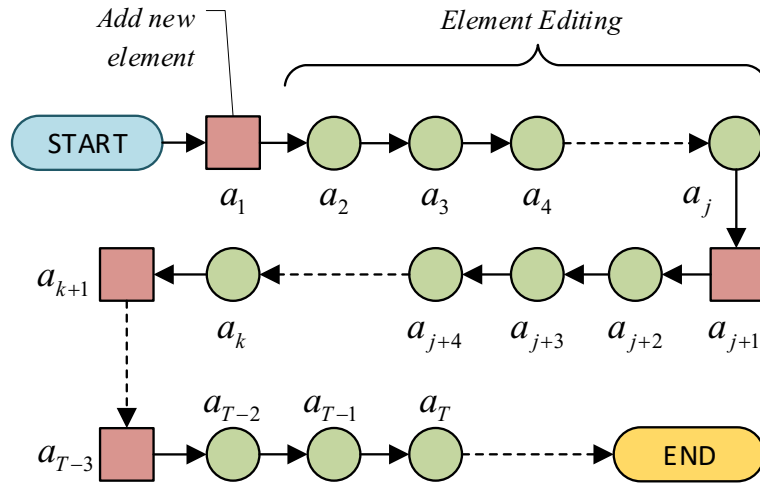


**FIGURE 3.3** The proposed framework of SvgAI. The editor is playing the role of the environment

As seen in common in RL settings, the proposed framework consists of two parts: the IA and the environment. As shown in [Figure 3.3](#), the editor is playing the role of the environment in this research. For every time step, the IA observes the state of the editor (step 1). Then, the IA processes the observed state and sends a new action to the editor (step 2). The editor executes the action as requested and sends a reward back to the agent (step 3). And, the process goes back to step 1.

#### 3.3.1 SVG Construction

In this research, our environment supports two fundamental shape elements (and with their transformations) that are square element and circle element for SVG image construction. [Figure 3.4](#) shows the process of SVG composition, where every episode starts with a blank canvas. During the process, the IA is either adding a new element into the working image or editing the most recently added element. The newly added element has a default presentation when it's firstly added. The default presentations of the circle and square elements are shown in sub-figure (a) and (g) of [Figure 3.11](#). As the agent keeps editing, the presentation of the element is to be updated. For example, sub-figure (l) in [Figure 3.11](#) is the presentation of a circle element after 400 steps of editing.



**FIGURE 3.4** The process of SVG composition: the IA keeps adding and editing new elements until the desired result is achieved.  $a_i$  is action given by the IA at time step  $t$ .  $T$  is the last time step in the episode.

### 3.3.2 Action Space

With the above-explained process, once a new element is added to the working SVG document, the old element is no longer editable. Thus, the consequence of adding new elements is more significant compared to editing them. Therefore, for Q-learning based training, it is crucial to distinguish the two sets of actions, set A and set B, and apply separated exploration policies during the training process. Otherwise, the model cannot converge. Table 3.1 lists all the actions of set A and B supported by the editor. Set A consists of actions that add new elements into the working SVG document, while set B consists of editing actions, i.e., element-shape manipulation actions.

**TABLE 3.1** List of Actions Supported By The SVG Editor.

Set	Action Id.	Description	
A	0, 1	Add circle/square element	
	2, 3, 4, 5	Move element left/right/up/down	
	6, 7, 8, 9	Compress/expand element horizontally/vertically	
	B	10, 11	Rotate the element clockwise/counter clockwise
		12, 13	Reduce/increase the line thickness
14, .., 21		Increase/decrease value of each channel of the line color (RGB $\alpha$ )	
22, .., 29		Increase/decrease value of each channel of the fill color (RGB $\alpha$ )	

### 3.3.3 State Observation

The state of the editor at any time step  $t$  consists of four components:

- Raster version of the SVG image being edited  $I_t$ .
- Target image  $Y$ , i.e. the image which the IA attempting to compose.
- Raster image of the element being edited  $I_t^*$ .
- An auxiliary vector that describes the state of the SVG element being edited. This auxiliary vector consists of parameters such as orientation, position, line thickness, line color, and fill color. All raster images are in the  $RGB\alpha$  format with the resolution of  $128 \times 128$  pixels.

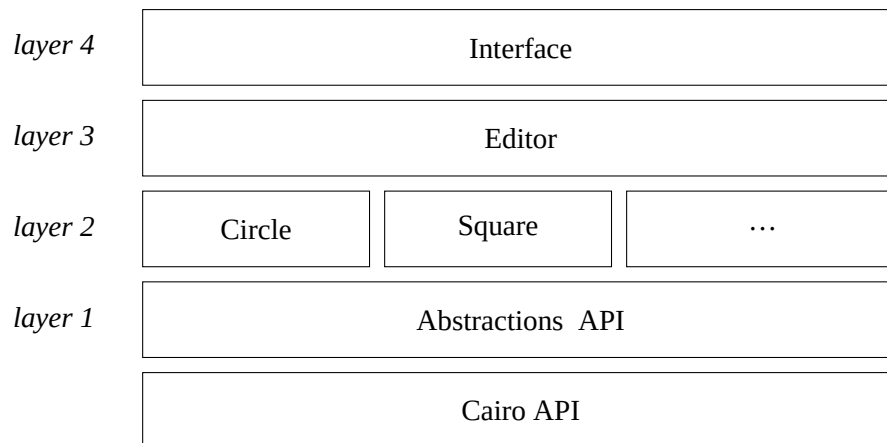
The first three components are stacked along channels dimension to form the image stack.

### 3.3.4 Implementation

#### 3.3.4.1 Back-end

The codebase of the research is in the Python programming language. There are numerous graphic libraries can be used as the rendering engine such as Python Imaging Library (PIL) [156] or Open source computer vision (OpenCV) [157]. However, Cairo [156] is used as the core library to implement the SVG editor due to the following advantages:

- It is an open-source programming application programming interface (API). It leads to cross-platform availability and continuous development support.
- Cairo is appealing technically with high-performance code and hardware acceleration.
- A large number of language binding available, including Python.
- Multiple back-ends support, including SVG and image buffers. Thus, it allows quick in-memory rendering during the drawing process and SVG image exporting when finished.
- The drawing model of Cairo naturally fits the purpose. It can be seen as a full-featured image editor without a graphical user interface (GUI). However, it does not provide functionality for modifying a drawn element. Hence, the implementation of the SVG editor must include abstraction layers that allow modification and communication interface with the IA.



**FIGURE 3.5** SVG Editor Environment Diagram

### 3.3.4.2 Structure

Figure 3.5 shows the diagram of the SVG editor environment. Built on top of Cairo API, the structure of the environment can be grouped into four layers.

**Layer 1** abstracts the necessary components of the API. One crucial component is called a Stepper. This component provides binary action to increase or decrease the continuous parameters of a shape object (layer 2 below). All numeric parameters of a shape are created from single or multiple Stepper components. Those parameters are angle, stroke thickness, color, gradient, coordinate, horizontal scale, to name a few.

The number of steps of each Stepper needs to be set based on the type of parameter and the domain of the parameter's value. For example, as shown in Listing 3.1, in our implementation, each channel of color is divided into 50 steps. A larger number of steps would lead to more difficulty in training since the change between each step is insignificant visually. However, too few numbers of steps also lead to poor performance due to value overshooting.

**Layer 2** abstracts the basic shapes supported by SVG schematic. These basic shapes abstraction essentially consists of multiple steppers. Each stepper record a parameter of the shape. For example, as shown in Listing 3.2, the circle element is abstracted by the Circle class. This class stores all basic parameters of a circle, including the coordinate of the center and the radius.

**Layer 3** abstracts the SVG editor. The objective of this editor is to manage multiple shapes object and providing a drawing API. All the shape objects are stored in a stack. The object positioned lower in the stack is overlaid by objects that are staying in a higher position.

### LISTING 3.1 Color Parameter Class Abstraction

```
1 class Color(MultiActors):
2     def __init__(self, name: str = "Color",
3                 r: int = 25, g: int = 25, b: int = 25, a: int = 49):
4         self.r = Stepper(name='R', _min=0., _max=1.,
5                          step=r, min_step=0, max_step=49, loop=False)
6         self.g = Stepper(name='G', _min=0., _max=1.,
7                          step=g, min_step=0, max_step=49, loop=False)
8         self.b = Stepper(name='B', _min=0., _max=1.,
9                          step=b, min_step=0, max_step=49, loop=False)
10        self.a = Stepper(name='A', _min=0., _max=1.,
11                        step=a, min_step=0, max_step=49, loop=False)
12
13        MultiActors.__init__(self, name, [self.r, self.g, self.b, self.a])
```

**Layer 4** interfaces with the IA by providing an action space that can be described by a one-hot vector. This action space is described in [Section 3.3.2](#). Because this action space is fixed, there are cases that a part of the action space results in no change to the editor status. For example, when an episode just started, because there is no shape element added to the image. Therefore, all action in action set B, as shown in [table 3.1](#), will not result in any change.

To make the environment complete, it provides additional functionality, such as setting the target image and reset the SVG editor in layer 3. The similarity between the target image and the image being edited is reported to the IA after every editing action.

#### 3.3.4.3 Drawing Operation

Even-though built on top of Cairo; the Cairo API is not used until the rendering process is initialized. Cairo abstracts a canvas by context object. This context is sequentially drawn by all shapes (layer 2 in [Section 3.3.4.2](#)) stored in the SVG editor (layer 3 in [Section 3.3.4.2](#)).

[Algorithm 2](#) describes this drawing operation. The drawing function shown on line 4 is corresponding to draw function on line 7 and line 19 of [Listing 3.2](#). These draw functions, in

---

#### Algorithm 2 Drawing Operation on Cairo's Context

---

```
1: procedure Draw(ctx)                                ▷ Draw all shape on blank context ctx
2:   for  $i \leftarrow 0, n$  do                            ▷  $n$  shape objects in the stack
3:      $shape \leftarrow s[i]$                                ▷ Get  $i$ -th shape object in stack  $s$ 
4:     shape draws on ctx
5:   end for
6:   return ctx
7: end procedure
```

---

**LISTING 3.2** Circle and Square Element Code Excerpt

```
1 class Circle(Elem):
2
3     def __init__(self, name: str = "Circle", radius: float = DEF.CIR.R):
4         Elem.__init__(self, name=name)
5         self.r = radius # Radius is fixed, no change
6
7     def draw_by(self, ctx: cairo.Context):
8         self._draw(ctx, ctx.arc, (0., 0., self.r, 0., 2 * math.pi))
9         # centerX, centerY, radius, startAngle, endAngle
10
11
12 class Square(Elem):
13     _S = DEF.Square
14
15     def __init__(self, name: str = "Square", edge: float = _S.edge):
16         Elem.__init__(self, name=name)
17         self.edge = edge # Edge is fixed, no change
18
19     def draw_by(self, ctx: cairo.Context):
20         self._draw(ctx, ctx.rectangle, (-1. * self.edge / 2,
21                                         -1. * self.edge / 2,
22                                         self.edge, self.edge))
23         # centerX, centerY, width, height
```

turn, called the draw function of the generic element that they are inherited from. The draw function of this generic element is shown in [Listing 3.3](#).

As shown in [Listing 3.3](#), this draw function also performs gradient filling, which is a commons among circle and square shape. The relative start and stop position of the gradient (lines 27 and 28) are fixed for the sake of simplicity.

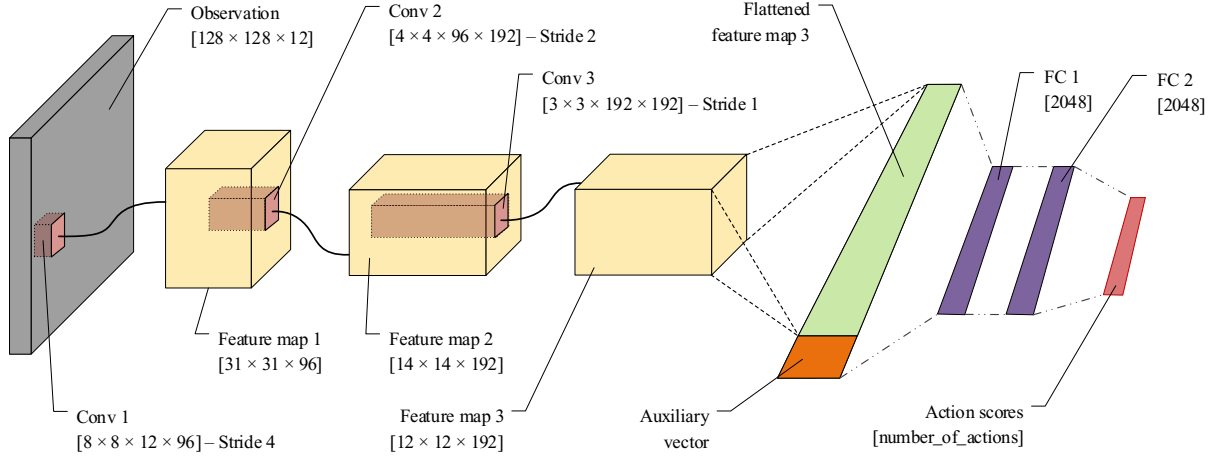
The final context is then exported into an in-memory buffer bitmap or SVG document. The in-memory bitmap is used during the training process for input as well as comparison. SVG document is used to evaluate other quality of the result, including size and number of elements.

In Cairo API, to export an image to a specific format, an appropriate context needs to be created. For reference, `cairo.Context(cairo.ImageSurface(cairo.FORMAT_ARGB32, w, h))` is used to export bitmap image and `cairo.Context(cairo.SVGSurface(path, w, h))` is used for SVG exporting in our implementation.

## 3.4 Model and Algorithms

### 3.4.1 Network Architecture

[Figure 3.6](#) shows the architecture of the proposed IA. The above-mentioned image stack



**FIGURE 3.6** The architecture of the IA. The stacked image explained in Section 3.3.3 is processed by the convolution layers. Auxiliary vector is concatenated with the output of the convolution layers before feeding to fully connected blocks at the end of the network.

is processed by the CNN blocks. These blocks produce a feature map with around  $27 \times 10^3$  parameters. The combination of this feature map and the auxiliary vector is then processed by a fully connected (FC) block to produce a score or probability for each action supported by the environment.

### 3.4.2 Error and Reward

With every action received from the agent, the score is calculated as follows:

$$g_t = \exp\left(-\frac{|I_t - Y|^2}{\sigma^2}\right) \quad (3.1)$$

Where  $g_t$  is the score at time step  $t$ , which describes the similarity between the raster version  $I_t$  of the working SVG document at time step  $t$  and the target image  $Y$ , and  $\sigma$  is the scaling factor. In our experiments,  $\sigma$  is set to 1. Thus, the domain of the score  $g_t$  is from 0 to 1, where 1 means perfect matching. Base on the score  $g_t$ , the reward is given to the IA as follows:

$$r_t = \begin{cases} 1 + g_t, & \text{if } g_t > g_{t-1}. \\ 0, & \text{otherwise.} \end{cases} \quad (3.2)$$

Where  $r_t$  is the immediate reward at time step  $t$ . Thus, the environment returns 0 reward when the action results in no improvement and returns a small reward ranging from 1 to 2 depending on the similarity between the result and the target image. Given  $v_t$  which is the number of SVG elements at time step  $t$ , the penalty  $p_t$  is given as follows:

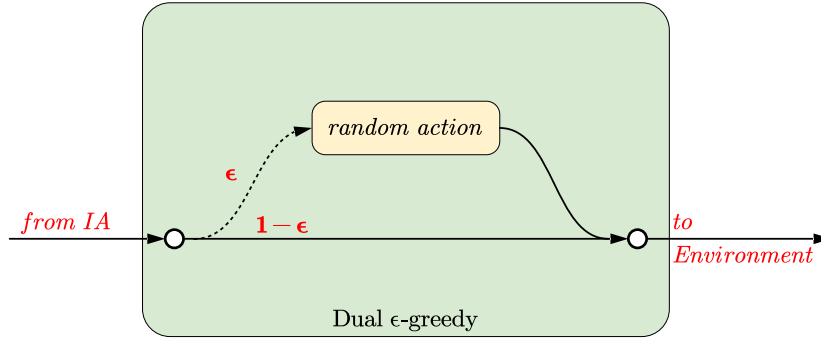
$$p_t = \begin{cases} -1, & \text{if } v_t > v_{t-1}. \\ 0, & \text{otherwise.} \end{cases} \quad (3.3)$$

Hence, the sum of discounted rewards (SDR)  $R_t$  at time step  $t$  is calculated as follows:

$$R_t = \sum_{i=t}^T (r_i + p_i) \gamma^{i-t} \quad (3.4)$$

Where  $\gamma$  is the discount factor, and  $T$  is the length of the episode. By giving a penalty for adding elements more than necessary, the IA is discouraged from performing actions in set A. However, this technique is only feasible with a controlled environment in which the structure of a state is known. The training process is expected to be slower without this penalty.

### 3.4.3 Q-learning and Exploration Policy



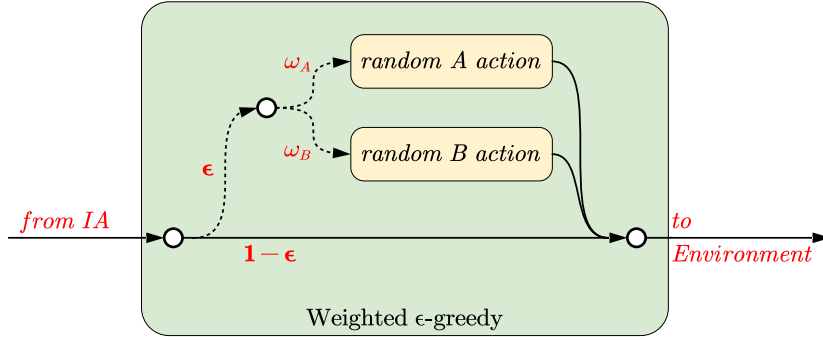
**FIGURE 3.7** The  $\epsilon$ -greedy exploration policy.

In our early attempts, the conventional  $\epsilon$ -greedy policy, as shown in Figure 3.7, was used. On average, with the action sets explained in Section 3.3.2, there was a new element added into the image for every 14 adjustments during the exploration process. It means, regardless of the value set to the exploration rate  $\epsilon$ , actions in set A were explored more than necessary. This bad exploration is most likely making the network to be trapped in sub-optimal solution [158]. The simplest solution is to apply lower weight for the actions in set A during the random exploration process. The probability for an action being performed randomly by the  $\epsilon$ -greedy policy is:

$$P(a_i) = \begin{cases} \epsilon \omega_A, & \text{if } a_i \in A. \\ \epsilon \omega_B, & \text{if } a_i \in B. \end{cases} \quad (3.5)$$

Where  $\omega_A$  and  $\omega_B$  are weights for the actions of set A and B. Figure 3.8 illustrates this weighted  $\epsilon$ -greedy policy.





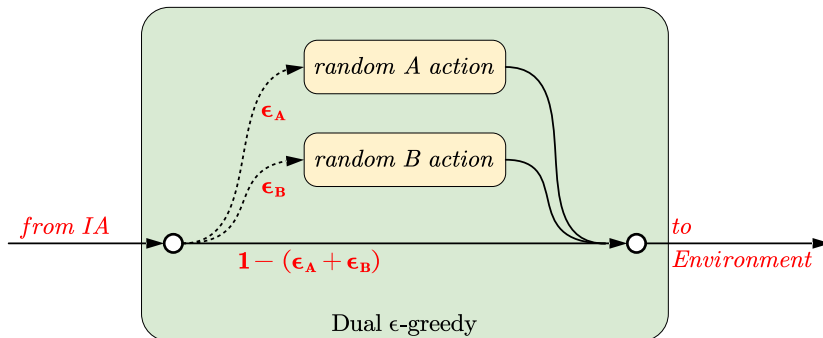
**FIGURE 3.8** The weighted  $\epsilon$ -greedy policy.

However, we observe that the agent trained by using this weighted  $\epsilon$ -greedy policy often inserts the incorrect element into the working image. A possible explanation for this problem is that the  $\epsilon$  value is already saturated when the agent does not learn the long-term reward of adding the right element yet.

Prolonging the exploration phase does not improve the result because the agent has to keep exploring more on element editing (i.e., action set B) in order to discover a better solution. Our approach to solving this problem is to apply the different reducing  $\epsilon$ -greedy policies (hereafter referred to as dual  $\epsilon$ -greedy policy) on each action set. In this setting, there are two independent reducing  $\epsilon$ -greedy polices applied for each action set:

$$P(a_i) = \begin{cases} \epsilon_A \omega_A, & \text{if } a_i \in A. \\ \epsilon_B \omega_B, & \text{if } a_i \in B. \end{cases} \quad (3.6)$$

Where  $\epsilon_A$  and  $\epsilon_B$  are weight values of two  $\epsilon$ -greedy policies for the action set A and action set B, respectively. In this way, the action set A and B can be independently explored. Figure 3.9 illustrates this dual  $\epsilon$ -greedy policy.



**FIGURE 3.9** The dual  $\epsilon$ -greedy policy.

Algorithm 3 shows the pseudo-code to train the agent using the Q-learning paradigm with dual  $\epsilon$ -greedy policy, where the random function has several forms (analogous to C++'s function

---

**Algorithm 3** Q-Learning Based Training Algorithm

---

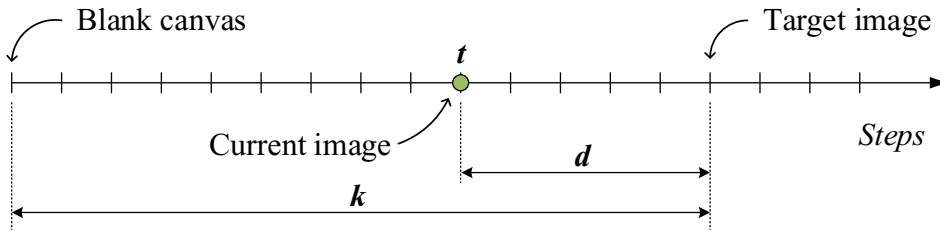
```
1: procedure Train( $\theta$ ) ▷ Optimize policy  $\theta$ 
2:    $Y \leftarrow$  random target image  $Y$  with random  $k$  ▷  $k$  is the difficulty of target
3:   for  $t \leftarrow 0, t_{\max}$  do ▷ Maximum time step  $t_{\max}$ 
4:      $gt \leftarrow gt + 1$  ▷ Global counter  $gt$ 
5:      $a_A \leftarrow \text{Rnd}(A)$ 
6:      $a_B \leftarrow \text{Rnd}(B)$ 
7:      $a_Q \leftarrow \max_a(Q_\theta(s_t, a))$ 
8:      $a_t \leftarrow \text{Rnd}(a_A, a_B, a_Q \mid p(a_A), p(a_B), q(a_Q))$ 
9:     Unroll episode with  $a_t$ 
10:     $M \leftarrow$  new episode
11:    if  $gt \bmod L = 0$  then ▷  $L$  is the training interval
12:      train mini batch taken from  $M$ 
13:    end if
14:    if episode end then
15:      break
16:    end if
17:  end for
18: end procedure
```

---

overloading). The function  $\text{Rnd}(A)$  and  $\text{Rnd}(B)$  randomly select an action in set  $A$  and  $B$ .  $\text{Rnd}(X|W)$  picks element from  $X$  with weight  $W$ . The difficulty  $k$  is the minimum number of steps to compose the target image from a blank canvas, as explained in Section 3.4.4.

### 3.4.4 Policy-Gradient

As mentioned in Section 2.3.2, in order to train the IA using policy-gradient,  $R_t$  has to be known. This value can be approximated for training. Thus, it makes the efficiency of the training process again depend on an external function. Another solution is to unroll the episodes fully. Monte Carlo method is commonly used for this purpose. However, this method is intractable in the environment with a large action space, as it requires an immense computational resource.



**FIGURE 3.10** The process of SVG composition: the IA keeps adding and editing new elements until the desired result is achieved.  $a_t$  is action given by the IA at time step  $t$ .  $T$  is the last time step in the episode.

To overcome this difficulty, we unroll the episode and train the IA reversibly. As shown in Figure 3.10, supposing that  $k$  is the minimum number of steps to compose the target image from a blank canvas, if an IA is trained to perform last  $d$  step from time step  $t$ , action at time step  $t - 1$  can be unrolled in a brute force way:

$$P(a_i | s_{t-1}) = \frac{[D(s_{t|a_j}, Y) = d]}{\sum_j [D(s_{t|a_j}, Y) = d]} \quad (3.7)$$

Where  $P(a_i | s_{t-1})$  is the probability of action  $a_i$  given the state  $s_{t-1}$  at time step  $t - 1$ ,  $D(s_{t|a_j}, Y)$  is the minimum number of steps for the agent to finish the episode given in the state  $s_{t|a_j}$  at time step  $t$  which is the result from action  $a_j$  from the last time step.

---

**Algorithm 4** Policy-Gradient Based Training Algorithm

---

```

1: procedure Train( $\theta$ )                                ▷ Optimize policy  $\theta$ 
2:   for  $d \leftarrow 1, k_{\max}$  do                        ▷ Upper limit  $k_{\max}$ 
3:     while  $\theta_d$  is not converged do                ▷ Train  $\theta$  to handle increasing difficulty
4:        $k \leftarrow U([d..k_{\max}])$                     ▷ Get random difficulty  $k$  in range  $[d..k_{\max}]$ 
5:        $t \leftarrow k - d$ 
6:        $episode \leftarrow \text{Unroll}(k)$                  ▷ Random episode at difficulty  $k$ 
7:       for  $i \leftarrow 0, n$  do                          ▷ Number of actions supported by editor  $n$ 
8:         if  $D(s_{s_t|a_i}, Y) = d$  then
9:            $M \leftarrow (s_{t-1}, a_i)$                 ▷ Memory replay  $M$ 
10:           $gt \leftarrow gt + 1$                           ▷ Global counter  $gt$ 
11:          if  $gt \bmod L = 0$  then
12:            train mini batch taken from  $M$ 
13:          end if
14:        end if
15:      end for
16:    end while
17:  end for
18: end procedure

```

---

Algorithm 4 shows the pseudo-code of policy-gradient based training. The agent is trained to work on incrementally difficult states. The difficulty of a state is measured by distance  $d$ . Once the policy network  $\theta_d$  for distance  $d$  is converged, it is then trained with more difficult distance  $d + 1$ .  $U(\cdot)$  is a uniform sampling function.

The primary disadvantage of this method is that it is only feasible with a controlled environment where the difficulty of the state can be calculated. However, it is useful in combination with techniques such as transfer learning when applying it for more complex data.

## 3.5 IA Setting and Evaluation

We train the IA by using both Q-leaning based and policy-gradient based methods. We evaluate the IA performance by comparing the similarity between generated SVG images and target images. Finally, we compare the quality between the SVG image produced by the proposed IA with that produced by popular R2V software. For more stable training process, we use Huber loss as follows:

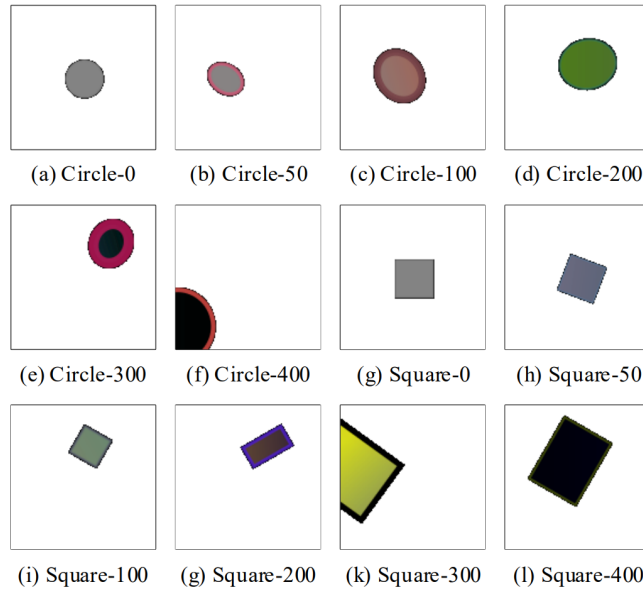
$$L(\alpha) = \begin{cases} \frac{1}{2}\alpha^2, & \text{if } |\alpha| \leq \delta. \\ \delta \left( |\alpha| - \frac{1}{2}\delta \right), & \text{otherwise.} \end{cases} \quad (3.8)$$

Where  $\alpha$  is the loss value,  $\delta$  is set to 1 in all of our experiments.

### 3.5.1 Setting

#### 3.5.1.1 Data Set

Works on image-processing based R2V conversion have many diverse objectives. As a result, they mostly use relatively plain and small data sets. The diversity in research objectives also leads to a lack of unified evaluation data set [49]. While the complexity of these data sets is suitable for this research, their modest size and heterogeneous properties are not adequate to be used to train deep neural network (DNN).



**FIGURE 3.11** Examples of target images generated for training and evaluation. The caption under each image shows the name of the element and the minimum number of steps ( $k$  as in Figure 3.10) required to compose that image from a blank canvas by using the editor.

With the above reasons, the training and the evaluation of the IA have been done on a randomly generated data set. It not only helps in avoiding the above-mentioned problems but also provides a controlled level of difficulty and the uniformity of the dataset. Target images are created by randomly generated SVG documents. These documents contain a single shape element with difficulty  $k$ . [Figure 3.11](#) shows examples of target images in format `<element>-<k>`.

### 3.5.1.2 Episode Termination

The episode is terminated if one of the following conditions is met:

- After a fixed time step  $t_{\max}$ . Since there is no further step after the termination, for Q-learning, the  $Q$  value as mentioned in (6) at the final step  $t_{\max}$  is  $Q_{\theta}(s_{t_{\max}}, a_{t_{\max}}) = r_{t_{\max}}$
- When  $g_t$  in 3.1 is greater than a certain threshold.

### 3.5.1.3 Frame Skip

Due to the computational demand of the IA, it is inefficient to let the IA perform an action at every time step. The popular solution is the frame skip [155], where the IA only interacts with the environment in every  $r$  steps. Therefore, once an action is decided by IA, it repeats  $r$  steps. A popular value of  $r$  in many tasks is 4, as it is usually a good trade-off between the performance and the training speed.

In this research, dismissing frameskip improves the performance of the IA because one action repeated several times makes the IA miss its target due to overshooting.

### 3.5.1.4 Parameter Update

The IA does not update its parameter at every time step, but once for every  $L$  experiences added in the replay memory. Thus, given the batch size of  $N$ , one experience is learned by the IA  $\frac{N}{L}$  times on average.

### 3.5.1.5 Parameter Settings

We have implemented the IA using the model proposed in [Section 3.4](#). We evaluate the performance of the IA trained by different training schemes:

- Policy-gradient
- Q-learning under different exploration policies

- Conventional  $\epsilon$ -greedy policy.
- Weighted  $\epsilon$ -greedy policy.
- Dual  $\epsilon$ -greedy policy.

**TABLE 3.2** Hyper-Parameter Setting for Experiments.

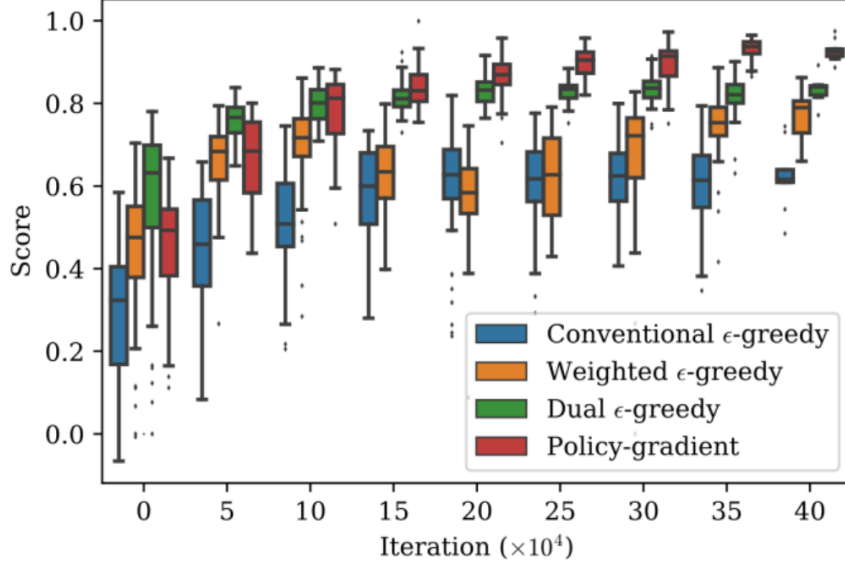
		Conventional	Weighted	Dual
Mini batch size		32	32	32
Upper limit $k_{\max}$		400	400	400
Train interval $L$		16	16	16
Discount factor $\gamma$		0.999	0.999	0.999
Memory replay size		$1e6$	$1e6$	$1e6$
$\epsilon$ -greedy	max-min	$1 - 0.1$	$1 - 0.1$	-
	start-end	$0 - 1e6$	$0 - 1e6$	-
$\epsilon_A$ -greedy	max-min	-	-	$1e3 - 0$
	start-end	-	-	$5e6 - 1e7$
$\epsilon_B$ -greedy	max-min	-	-	$0.999 - 0.1$
	start-end	-	-	$0 - 1e6$
$W_A$		1	$1e - 3$	1
$W_B$		1	$999e - 3$	1

Table 3.2 shows the hyper-parameters used for the experiments. Policy-gradient training experiments share the first 5 parameters with other experiments.

## 3.5.2 Evaluation

### 3.5.2.1 Performance

Figure 3.12 shows the distribution of the evaluation scores of the IA under different training schemes. Similar to the training configuration, the target images used for evaluation are generated with random difficulty  $k$  given that  $k \leq 400$ . The box plot describes the distribution of evaluation scores by  $5 \times 10^4$  iterations interval from 0 to  $40 \times 10^4$ . For each scheme, we train the IA 5 times. Each time, we evaluate the IA after every 100 training iteration and collect evaluation score calculated using equation 3.1. Thus, each box describes the distribution of 500 evaluation scores. The colored box indicates interquartile range (IQR). The horizontal line within the box indicates the median score, and the extended bar indicates the maximum and minimum scores. Dots indicate outliers. Our proposed dual  $\epsilon$ -greedy policy shows not only significant performance gain but also highly stable compared to conventional  $\epsilon$ -greedy policy



**FIGURE 3.12** Evaluation score distribution of the IA throughout the training process under different training scheme.

and weighted  $\epsilon$ -greedy policy during the training process. The IA trained by policy-gradient achieves the best result and the high stability when the number of iterations is more than or equal to  $15 \times 10^4$ .

### 3.5.2.2 Accuracy

With dual  $\epsilon$ -greedy policy, our trained IA favors adding circle elements and achieves higher scores by editing circle elements in general. To further analyze this observation, we evaluate each trained IAs for 500 episodes with the same setting used in Section 3.5.2.1 and drill down the evaluation results. In general, evaluation results can be divided into two sets:

**Circle set** consists of episodes with target images that contain circle elements only (250 episodes for each IA).

**Square set** consists of episodes with target images that contain the square elements only (250 episodes for each IA).

With each set, we analyze the performance of the IA on action set A and set B separately. The IA’s performance on action set A is measured by the number of episodes where a correct shape element inserted into the working SVG document over the total number of episodes in the set. Correctly inserting a shape is important because further editing the incorrect element results in a large number of bad experiences that negatively affects the IA’s policy network. Table 3.3 shows the performance on action set A of the IAs trained under different training schemes.

**TABLE 3.3** IA Performance on Action Set A

	Circle set	Square set
Policy-gradient	<b>0.99</b>	<b>0.95</b>
Q-learning		
Dual $\epsilon$ -greedy	0.94	0.76
Weighted $\epsilon$ -greedy	0.67	0.58
Conventional $\epsilon$ -greedy	0.53	0.61

To evaluate the IA’s performance on action set B, we again evaluate each IA for 500 episodes with an element already inserted. The evaluation results are divided into two sets:

**Set 1** consists of episodes with target images that contain the same element with the pre-inserted element (250 episodes each IA).

**Set 2** consists of episodes with target images that contain an element that is different from the pre-inserted element (250 episodes for each IA).

**TABLE 3.4** IA Performance on Action Set B

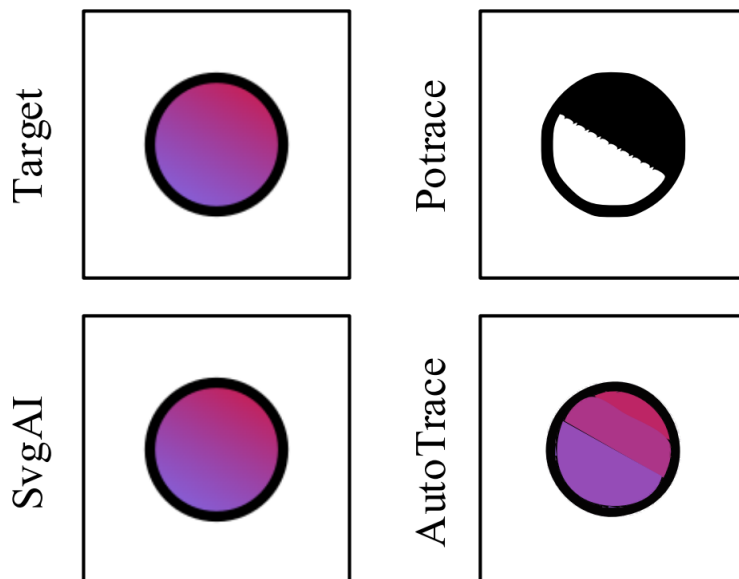
IA	Pre-inserted	Target Image	
		Circle	Square
Policy-gradient	Circle	<b>0.97</b>	0.65
	Square	0.71	<b>0.94</b>
Q-learning	Dual $\epsilon$ -greedy	Circle	0.93
		Square	<b>0.72</b>
	Weighted $\epsilon$ -greedy	Circle	0.81
		Square	0.67
Conventional $\epsilon$ -greedy	Circle	0.59	
	Square	0.61	

Table 3.4 shows the IA’s performance on action set B. The evaluation scores of set 1 are in the gray background. For this set, the performance of IAs trained by policy-gradient is better than the IAs trained by Q-learning. The evaluation scores of set 2 are in white background. Interestingly, for set 2, even with the wrong element pre-inserted, the performance of all the trained IAs is over 0.5. This reflects the fact that all the IAs are trained with a sustainable amount of episodes in which the wrong element is inserted. This result is also correlated to action set A performance shown in Table 3.3: Because the IAs trained by policy-gradient have high accuracy for action set A, they rarely experience the training episodes where the wrong



element is added. Thus, on set 2, their performance is even lower than the performance of IAs trained by dual  $\epsilon$ -greedy policy (bold italic v.s. italic on Table 3.4).

### 3.5.2.3 SVG Quality



**FIGURE 3.13** Comparison between SVG images produced by SvgAI, Potrace and AutoTrace. SvgAI trained by Q-learning and policy-gradient produces identical result for this example.

We visually compare the SVG image produced by our trained IA with two popular open source and commercial R2V solutions: Potrace and AutoTrace [159]. Figure 3.13 shows the comparison between the outputs. As shown in the figure, Potrace not only has a problem with color quantization but also results in a distorted circle. On the other hand, AutoTrace produces a much better result. However, the linear gradient fill has been converted into color blobs. Without any manual configurations before the conversion/drawing process, our IA produces a significantly better result.

**TABLE 3.5** Average SVG Size Comparison

	Target	SvgAI	Potrace	AutoTrace
File Size	5.8KB	<b>852B</b>	1.4KB	15.5KB
Node Count	-	<b>1.4</b>	2.7	174
Color	$\geq 16 \times 10^6$	$\geq 16 \times 10^6$	2	10

Not only visually better but the SVG images produced by our IA are also smaller both in file size and node counts. As shown in Table 3.5, SVG images produced by our IA are 40%

smaller in size and 63% smaller in node counts than the second-best solution in average over 100 images where each one contains a single element, i.e., a circle or a square.

LISTING 3.3 Code Excerpt of Generic Element's Drawing

```
1 class Elem(MultiActors):
2     def __init__(self, name: str = "Elem"):
3         """ Initiate SVG Element instance
4
5         """
6         self.center = Point('Center')
7         self.scale = Scale()
8         self.angle = Angle()
9         self.lineWidth = LineWidth()
10        self.lineColor = Color('LineColor')
11        self.fill = Gradient("FillGradient")
12
13        attrs = [self.center, self.scale, self.angle,
14                self.lineWidth, self.lineColor, self.fill]
15        MultiActors.__init__(self, name, attrs)
16
17    def transform(self, ctx: cairo.Context):
18        ctx.translate(*self.center.tuple)
19        ctx.rotate(self.angle.value)
20        ctx.scale(*self.scale.tuple)
21        return self
22
23    def _draw(self, ctx, func, arg):
24        ctx.save()
25        self.transform(ctx)
26        func(*arg) # Rect, act, etc..
27        ctx.restore()
28
29        # RENDER FILL GRADIENT
30
31        rad = self.fill.angle.value
32        points = np.array([[0., 0.],
33                          [1., 0.]])
34        rot = np.array([[math.cos(rad), -math.sin(rad)],
35                       [math.sin(rad), math.cos(rad)]])
36        points = np.dot(points, rot)
37        points[:, 0] = points[:, 0] - points[:, 0].mean() + 0.5
38        points[:, 1] = points[:, 1] - points[:, 1].mean() + 0.5
39
40        grad = cairo.LinearGradient(points[0, 0], points[0, 1],
41                                   points[1, 0], points[1, 1])
42
43        grad.add_color_stop_rgba(0., *self.fill.startColor.tuple)
44        grad.add_color_stop_rgba(1., *self.fill.endColor.tuple)
45
46        # END RENDER FILL GRADIENT
47
48        ctx.set_source(grad)
49        ctx.fill_preserve()
50        ctx.set_source_rgba(*self.lineColor.tuple)
51        ctx.set_line_width(self.lineWidth.value)
52        ctx.stroke()
53        return self
```



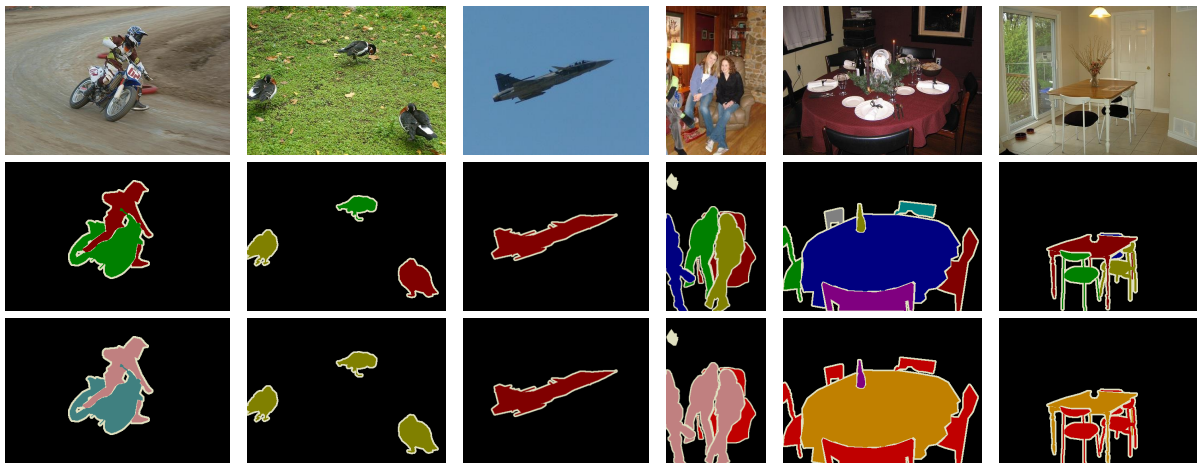
## CHAPTER 4

# Semantic Segmentation for Street Fashion Photos<sup>1</sup>

## 4.1 Introduction

### 4.1.1 Semantic Segmentation

Semantic segmentation has been a challenge in the field of computer vision (CV) even prior to the deep learning (DL) era. Classic object detection and classification requires only bounding boxes and classification of the object. Semantic segmentation requires individual pixels to be mapped into a predefined class.



**FIGURE 4.1** Sample of VOC2007 dataset on semantic segmentation [160]. The first row shows input photos. The second row shows object segmentation ground truth. The third row shows class segmentation ground truth.

<sup>1</sup>This chapter refers to the author's publication [11].

Figure 4.1 shows semantic segmentation samples from Pascal VOC2007 dataset [160]. As shown in the figure, besides class segmentation, there is also object segmentation. In object segmentation, each pixel required not only to be assigned into a class but also to an object instance of a class.

Recently, in 2019, Kirillov et al. proposed a new class of semantic segmentation called panoptic segmentation [161]. In this new class of semantic segmentation, both foreground and background of the photo are required to be segmented. For background segmentation, only class segmentation is required. However, object segmentation level is required for foreground segmentation.

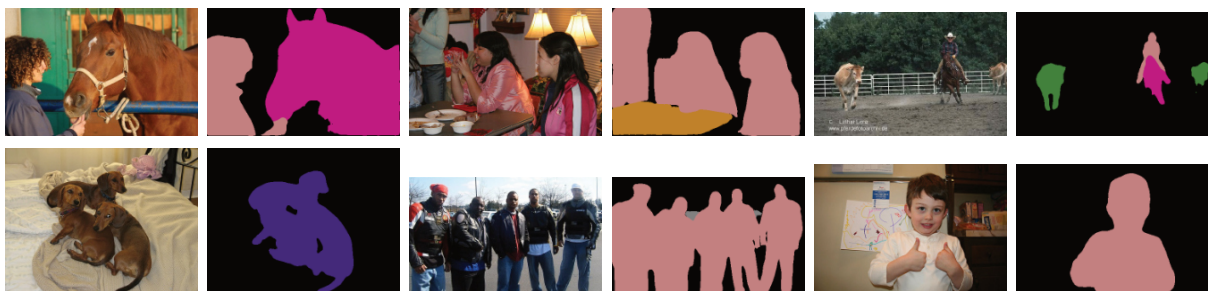


FIGURE 4.2 Class semantic segmentation outputs of deeplabv3+ [18].

Prior to the deep learning era, the state-of-the-art works have been based on Texton Forest [56] and conditional random field (CRF) [57]. Like other branches of CV, the performances of the classic models are limited compared to new deep neural network (DNN) models. Figure 4.2 shows the output of Deeplabv3+ [18] on Pascal VOC2012 dataset. Deeplabv3+ [18], together with SegNet [17], U-Net [65], and PSPNet [19] are popular and considered to be state-of-the-art. MSCOCO [66], CityScapes [67] and ADE20K [68] are popular dataset for training and benchmarking semantic segmentation works.

#### 4.1.2 Semantic Segmentation for Street Fashion Photos

In 2018, Zheng et al. published ModaNet [12]. ModaNet is the first large-scale street fashion data set with pixel-level annotation. This data set consists of 55,176 fully annotated images, where 52,377 images are for training, and the remaining 2,799 images are used for validation. With such a large dataset, it is possible to retrain ever-proposed architectures using ModaNet.

Figure 4.3 shows samples from our custom street fashion data set. This set of samples is highly resemblance to the samples from ModaNet. The color for each class in this figure is shown in Table 4.2 together with other related statistics.

We interested in a semantic segmentation system for street fashion photos because of its practicality. This system is crucial for a fashion recommender system. Moreover, such a rec-



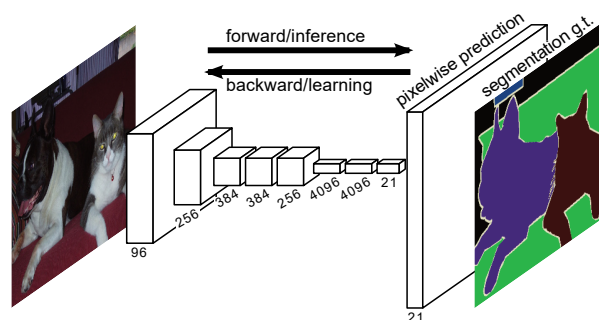
**FIGURE 4.3** Samples from our custom street fashion data set. In the top row, original images are shown, and in the bottom row, corresponding segmentation ground truths are shown. The class names for each color are shown in Table 4.2. Photos are public domain works downloaded from Pexels.com, and labels are manually annotated by the authors.

ommender system has long been an interest of the general consumer. This interest is also reflected by the number of investments in the field. According to [6], fashion and retailing was the second most popular AI startup investment in the United States in 2019. In Europe, it was the most popular investment category.

A semantic segmentation system for street fashion photos is more meaningful to be run on mobile devices. However, ever-proposed networks such as deeplabv3+ are computational demanding. In this chapter, we propose a lightweight yet high-performance semantic segmentation model.

## 4.2 Related Works

### 4.2.1 Fully Convolutional Neural Network



**FIGURE 4.4** Structure of Fully Convolutional Neural Network [58].

Early works on this topic mostly adopt the straight network design. The first proposed model is Fully convolutional neural network (FCN) [58]. Figure 4.4 shows the structure of FCN. The most important contribution of FCN is converting the fully connected classification layers of image classification networks into a  $1 \times 1$  (i.e., point-wise) convolutional layers to produce pixel-level segmentation prediction. Hence, it can be implemented on top of the ever-proposed classification models such as GoogLeNet [13], VGG [63] and ResNet [64]. The authors of FCN have found that their proposal works best by using VGG-16 as the network base.

## 4.2.2 PSPNet

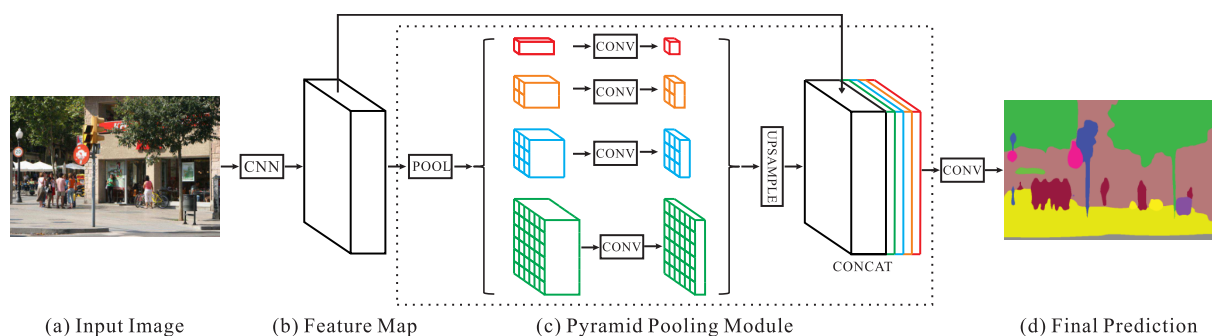


FIGURE 4.5 Structure of PSPNet [19].

Figure 4.5 shows the structure of PSPNet [19]. PSPNet introduces a spatial pyramid pooling scheme, which results in better context-awareness in the final result. In this pyramid pooling scheme, features maps from different layers of the base network are resized and concatenated. The concatenated feature map is then used as an input for a point-wise CNN to produce segmentation result.

## 4.2.3 SegNet

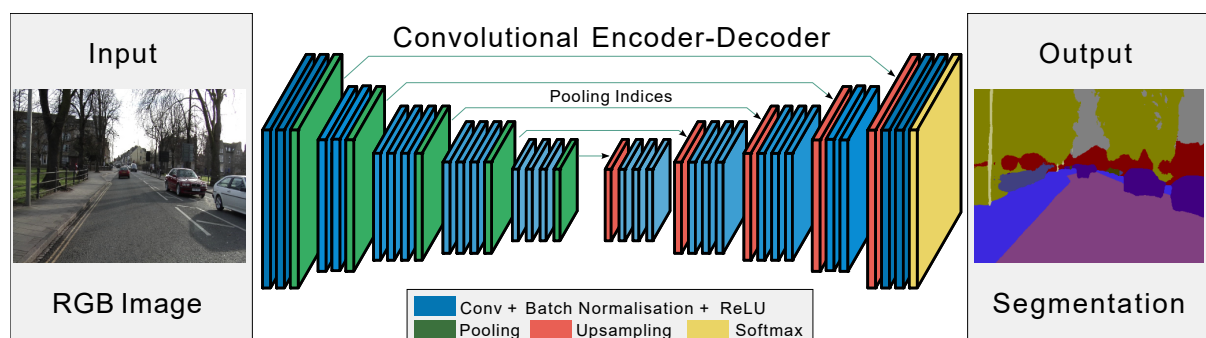


FIGURE 4.6 Structure of SegNet [17].



Later works on the topic mostly utilize the encoder-decoder structure. Models following this approach is usually yielding better performance. Popular models in this category include SegNet [17]. SegNet is a CNN based autoencoder. Figure 4.6 shows the structure of SegNet. It utilizes the indices from 2D max-pooling layers in the encoder to upscale the feature map using unpool layers in the decoder.

#### 4.2.4 U-Net

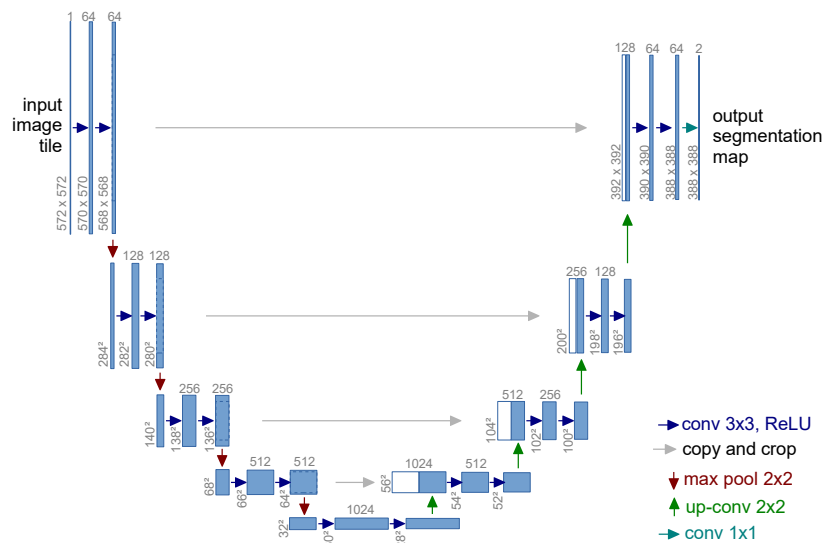


FIGURE 4.7 Structure of U-Net [65].

Figure 4.7 shows the structure of U-Net [65]. As shown in the figure, the structure of U-Net is similar to the structure of SegNet. However, U-Net implements skip connections between the corresponding encoder and decoder blocks. Introduced in 2015, U-Net is one of the earliest works on this topic. Nevertheless, its simplicity in design and high performance keep it popular even in 2020.

#### 4.2.5 DeepLabv3+

In [59], Yu and Koltun propose both dilated convolutional neural network (CNN) for semantic segmentation and a reference network design. Dilated CNN allows the deeper layers of the network to capture the context without losing resolution. The main drawback of this design is the high demand for computational resources because the feature map is rarely down-sampled.

DeepLabv3+ [18] combines all of the above approaches and achieves state-of-the-art performance in many benchmarks. Figure 4.8 shows the structure of DeepLabv3+.

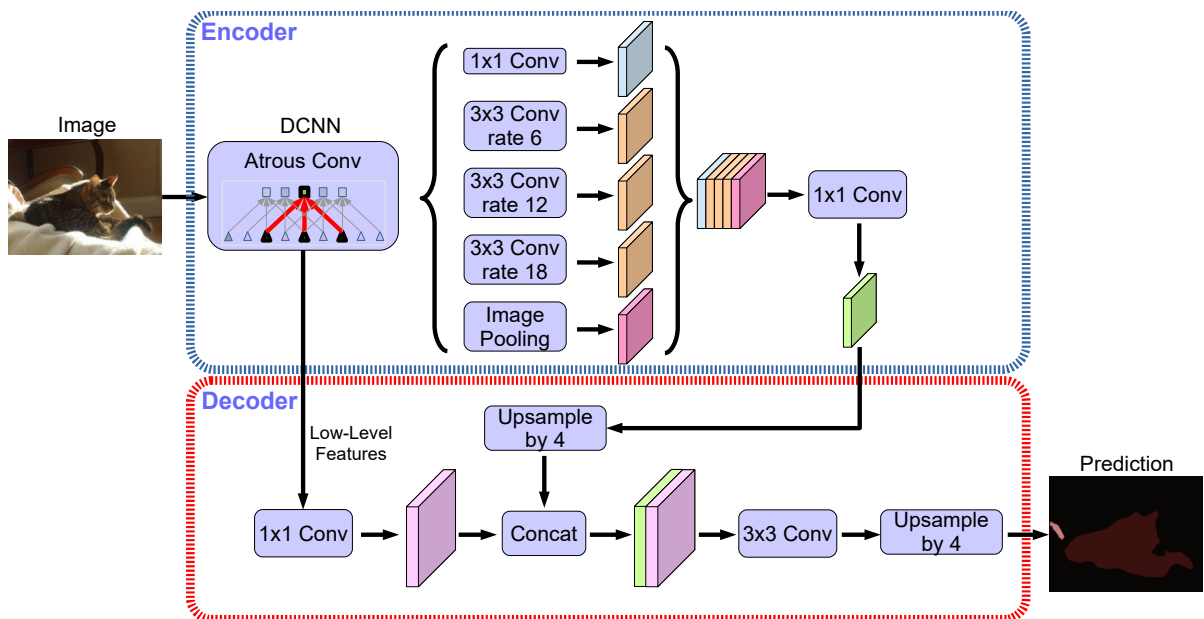


FIGURE 4.8 Structure of DeepLabv3+ [18].

## 4.2.6 Auxiliary Losses

As networks become deeper, new challenges arise. One of the most challenging problems is vanishing gradient [162]. In this problem, the gradient becomes too small in the layers being far away from the training loss function.

At first, auxiliary losses are commonly used to overcome the problem. For example, in GoogLeNet [13], besides the main softmax classification loss at the end of the network, another two similar classification losses are added into the middle of the network. Thus, the weights of early blocks are learned mostly by gradient propagated from auxiliary losses. In research on GANs [163], besides the usual real or fake discrimination, Chen et al. propose an auxiliary loss to discriminate the orientation of the input and output pairs to produce a more robust model. Undoubtedly, selecting the type of auxiliary objectives and their position greatly influences the performance of the network. The auxiliary training objectives also depict the type of features learned by the network. Thus, it does not guarantee that the best feature would be learned.

Another solution to the gradient vanishing problem is using skip connections [121, 127]. In [164], skip connections are used to patch feature maps from early blocks to deeper blocks of the encoder. In [165], ResBlocks [64] are used to replace the conventional CNN blocks in both encoder and decoder, resulting in a very deep encoder-decoder based network.

Even though skip connection has become more popular due to its simplicity, it is not the replacement for auxiliary loss. Perhaps, they can be complements to each other. In [19], Zhao et al. conduct an ablation study for auxiliary loss on ResNet [64] based FCN [58]. By adding an auxiliary loss after the res4b22 residue block and weighted it appropriately, the network

performance is gained by 0.94% on pixel accuracy. In [166], multiple spatially scaled versions of training labels are used as auxiliary training objectives.

## 4.3 Proposals

The work of Street Fashion Semantic Segmentation (SFSS) is done with three main contributions. Firstly, the network structure with a unique lightweight design that enables high segmentation performance. Secondly, the label pooling feature to be used as an auxiliary training objective. And lastly, a guided training process with multiple training objectives that enhances training effectiveness.

### 4.3.1 Network Design

#### 4.3.1.1 Problems

Two common problems of semantic segmentation are category confusion and inconspicuous segmentation [19]. Despite efforts to tackle the problems in previous researches such as [18] and [19], the problems still occur on street fashion photos as shown in Figure 4.18 in Section 4.4.6.

In category confusion, the models fail to identify the correct class of the whole segment. For example, PSPNet fails to identify the outer-wear in image (n). Moreover, with image (a) and (b), all the models recognize boots as ordinary footwear. Another example of this problem is the segmentation of image (k) by DeepLabv3+. We observe that this problem usually happens with networks that have high context-awareness.

When the above-mentioned problem is limited to local areas, it creates inconspicuous segmentation. For example, with input image (i), SegNet detects parts of the dress as top-wear and outer-wear. With image (f), PSPNet frequently confuses between pants and skirts. Thus, it results in segmentation with a considerable amount of noise.

PSPNet [19] deliberately addresses this problem by proposing the spatial pyramid pooling module (PPM). This module is expected to increase the size of the receptive field of the network. PPM is also adopted in [18]. However, it appears that the receptive field is still limited to the street fashion problem. A possible reason is that the PPM operates on the feature map produced by a CNN head. Thus, information is already lost during the process, and the important information may not be produced simply by pooling the feature map.

**TABLE 4.1** Network Parameters

Block	Output	Filters	Kernel	St. <sup>a</sup>	Pd. <sup>b</sup>	Ac./Op. <sup>c</sup>
A	224 × 224	32	5 × 5	1	2	ReLU
		64	3 × 3	1	1	ReLU
		32	1 × 1	1	0	ReLU
B <sub>1</sub>	112 × 112	64 3	3 × 3 3 × 3	1 1	1 1	ReLU Sigmoid
B <sub>2</sub>	56 × 56	— — —	— — —	— — —	— — —	— — —
B <sub>3</sub>	28 × 28	— — —	— — —	— — —	— — —	— — —
B <sub>4</sub>	14 × 14	— — —	— — —	— — —	— — —	— — —
E <sub>0</sub>	112 × 112	64	4 × 4	2	1	ReLU
		128	3 × 3	1	1	ReLU
		64	1 × 1	1	0	ReLU
E <sub>1</sub>	56 × 56	128	— — —	— — —	— — —	— — —
E <sub>2</sub>	28 × 28	256	— — —	— — —	— — —	— — —
E <sub>3</sub>	14 × 14	512	— — —	— — —	— — —	— — —
E <sub>4</sub>	7 × 7	1024	— — —	— — —	— — —	— — —
E <sub>5</sub>	3 × 3	1024	3 × 3	3	1	ReLU
		2048	3 × 3	1	1	ReLU
		1024	1 × 1	1	0	ReLU
E <sub>6</sub>	1 × 1	1024	3 × 3	1	0	ReLU
		2048	1 × 1	1	0	ReLU
		1024	1 × 1	1	0	ReLU
C <sub>0</sub>	112 × 112	128	3 × 3	1	1	ReLU
	224 × 224	1	2 × 2	2	0	UnPool
C <sub>1</sub>	112 × 112	— — —	— — —	— — —	— — —	— — —
C <sub>2</sub>	56 × 56	— — —	— — —	— — —	— — —	— — —
C <sub>3</sub>	28 × 28	— — —	— — —	— — —	— — —	— — —
C <sub>4</sub>	14 × 14	— — —	— — —	— — —	— — —	— — —
C <sub>5</sub>	3 × 3	128	3 × 3	1	1	ReLU

Block	Output	Filters	Kernel	St. <sup>a</sup>	Pd. <sup>b</sup>	Ac./Op. <sup>c</sup>
C <sub>5</sub>	7 × 7	1	3 × 3	3	1	UnPool
C <sub>6</sub>	1 × 1	128	1 × 1	1	0	ReLU
	3 × 3	1	3 × 3	3	0	UnPool
D <sub>0</sub>	112 × 112	128 14	3 × 3 3 × 3	1 1	1 1	ReLU Sigmoid
	224 × 224	1	2 × 2	2	0	UnPool
D <sub>1</sub>	112 × 112	— — —	— — —	— — —	— — —	— — —
D <sub>2</sub>	56 × 56	— — —	— — —	— — —	— — —	— — —
D <sub>3</sub>	28 × 28	— — —	— — —	— — —	— — —	— — —
D <sub>4</sub>	14 × 14	— — —	— — —	— — —	— — —	— — —
D <sub>5</sub>	3 × 3	128 14	3 × 3 3 × 3	1 1	1 1	ReLU Sigmoid
	7 × 7	1	3 × 3	3	1	UnPool
D <sub>6</sub>	1 × 1	128 14	1 × 1 1 × 1	1 1	0 0	ReLU Sigmoid
	3 × 3	1	3 × 3	1	0	UnPool
P <sub>0</sub>	56 × 56	1	2 × 2	2	0	MaxPool
P <sub>1</sub>	28 × 28	— — —	— — —	— — —	— — —	— — —
P <sub>2</sub>	14 × 14	— — —	— — —	— — —	— — —	— — —
P <sub>3</sub>	7 × 7	— — —	— — —	— — —	— — —	— — —
P <sub>4</sub>	3 × 3	— — —	3 × 3	3	1	— — —
P <sub>5</sub>	1 × 1	— — —	3 × 3	1	1	— — —
L <sub>0</sub>	224 × 224	128 14	3 × 3 3 × 3	1 1	1 1	ReLU Softmax
L <sub>1</sub>	112 × 112	— — —	— — —	— — —	— — —	— — —
L <sub>2</sub>	56 × 56	— — —	— — —	— — —	— — —	— — —
L <sub>3</sub>	28 × 28	— — —	— — —	— — —	— — —	— — —
L <sub>4</sub>	14 × 14	— — —	— — —	— — —	— — —	— — —

<sup>a</sup>Stride, <sup>b</sup>Padding, <sup>c</sup>Activation/Operation

### 4.3.1.2 Direction

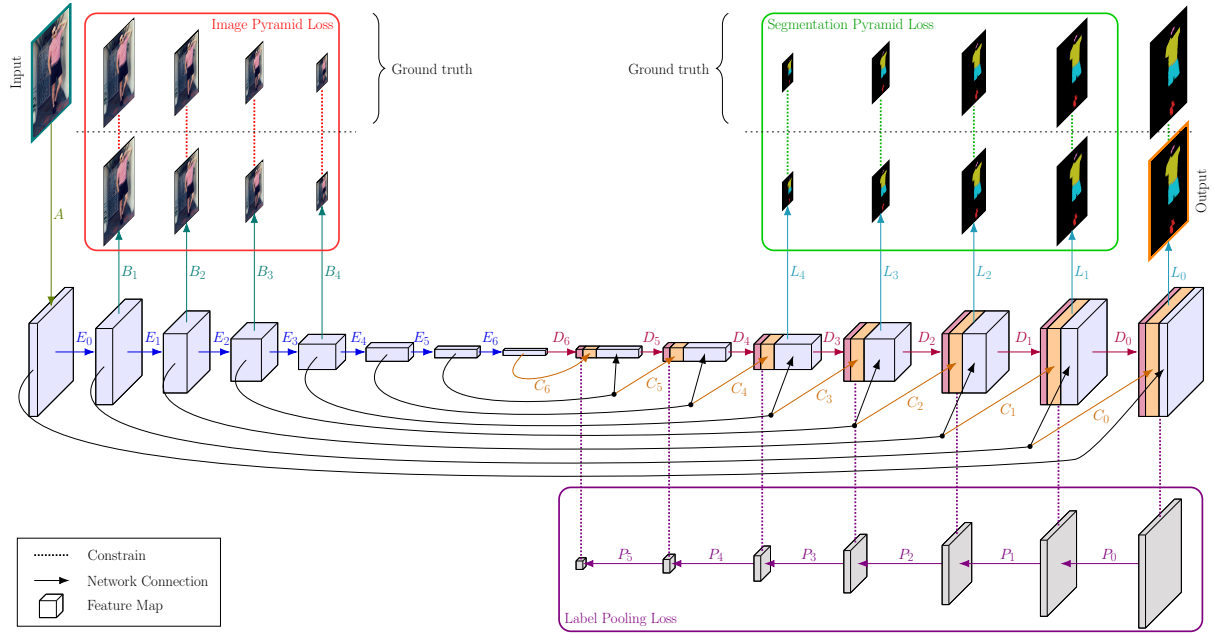
We observe that, for street fashion photos, the above-mentioned problems can be eliminated by knowing whatever a type of apparels is presented in the whole image. For example, in image (d) as shown in Figure 4.18, there are only two types of apparels presented in the image that are dress and pants (a small dark gray area under the model’s left arm as in the ground truth image). Thus, if a network only considers dress and pants for segmentation result, the problem of class confusion and inconspicuous would greatly be reduced.

On the one hand, it is uncomplicated to create a separated model to detect whatever the type of clothes is presented in an image. On the other hand, it is not efficient to create and train separated networks to solve a single problem. Therefore, we merge two types of networks into one and further extend the concept of apparel detector to all of the scales.

### 4.3.1.3 Implementation

Based on the encoder-decoder structure, we first set the length of the network so that the feature map at the end of the decoder is  $1 \times 1$ . It is to ensure the high context awareness of the network. Secondly, at every scale of the decoder, we expect the network to produce a prediction to indicate whatever the type of clothes is presented in the receptive field of the corresponding pixel of the feature map, i.e., the network first detects the presence of apparel type over the whole image, and then refines it until reaching the required resolution. Ground truth for such prediction can be produced by applying 2D max-pooling on the one-hot vector form of the original ground truth. This process is explained in detail in Section 4.3.2.

### 4.3.1.4 Network Structure





**FIGURE 4.9** Overview of the Proposed Network Structure with all three auxiliary losses. The three auxiliary losses are explained in detail in Section 4.3.3. Ground truth for **Image Pyramid Loss** and **Segmentation Pyramid Loss** are scaled versions of the image and the ground truth segmentation. However, in **Label Pooling Loss**, the initial ground truth on the bottom right of the figure is the one-hot vector version of the ground truth segmentation. This one-hot version of segmentation is then progressively scaled down by  $P_{[0..5]}$ . This label pool feature is explained in detail in Section 4.3.2. Moreover, in **Label Pooling Loss**, constraints (.....) are made only between label pool feature maps (Ⓜ) and output of decoders (Ⓜ). Network connections that are not necessary to generate output  from input  are ignored during inference. The detailed configuration of the whole network is described in Table 4.1.

Figure 4.9 shows the structure of our proposed network. It comprises two main parts: encoder and decoder. Both the encoder and decoder parts consist of 7 CNN blocks ( $E_i$  and  $D_i$  blocks where  $i \in [0..6]$ ). Feature map is downscaled every time it is processed by an encoder

block, and correspondingly upsampled every time it is processed by a decoder block. We organize this network into 7 different levels based on 7 different scales of the feature maps.

Similar to U-Net, skip connections with identity function are implemented between encoder and decoder blocks of the same level (black arrows  $\rightarrow$  as in Figure 4.9). However, in our proposal, the feature map produced by an encoder also leaks into the next level of the decoder. To adapt the feature map into the larger scale, we employ CNN - 2D unpooling blocks  $C_i$ . In our network, encoder blocks scale down the feature map by utilizing CNN with stride 2 instead of using 2D max-pooling operations. Thus, different from SegNet [17], the 2D unpool layers in our network do not utilize the pooling indices.

As mentioned, in this network, we expect decoder blocks to produce the prediction on the presence of a class within the whole image and then gradually refine the prediction result. Therefore, all the decoder blocks have the same design that output only 14 channels feature map, which is the number of segmentation classes of ModaNet (13 classes plus background as shown in Table 4.1).

Element-wise sigmoid function is used as activation function for  $D_i$  blocks as follows.

$$d_i = \frac{1}{1 + \exp(-d'_i)} \quad (4.1)$$

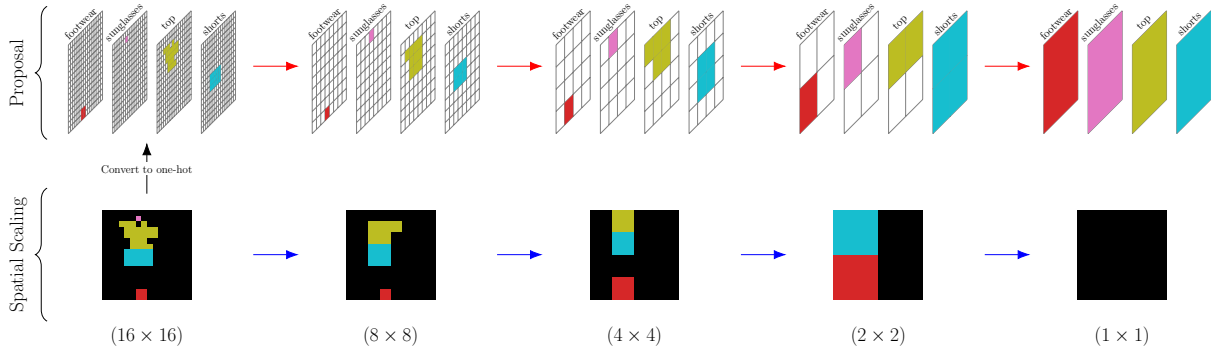
Where  $d_i$  is the output of  $D_i$  block, and  $d'_i$  is the pre-activation value of  $D_i$ .

Segmentation prediction is produced by  $L_0$  block. In this network, besides  $L_0$ , there are another 4  $L_i$  blocks where  $i \in [1..4]$ . These blocks produce smaller-scale versions of the segmentation prediction. In general, the scale of the prediction produced by  $L_i$  block is  $2^{-i}$ . The input of  $L_i$  block is the concatenation of feature maps output from  $C_i$ ,  $D_i$  and  $E_i$  blocks. Pixel-wise softmax is used to produce the output of  $L_i$  blocks as follows.

$$l_{ij} = \frac{\exp(l'_{ij})}{\sum_{k=1}^K \exp(l'_{ik})} \quad (4.2)$$

Where  $l_{ij}$  denotes the value of channel  $j$  of the feature map produced by  $L_i$ ,  $l'_{ij}$  denotes the pre-activation value of  $l_{ij}$ , and  $K$  denotes the total number of channels which also is the number of segmentation classes.

From level 1 to level 4, different scales of the input image are reconstructed by  $B_i$  blocks where  $i$  is the level number. The input of  $B_i$  block is the feature map  $e_{i-1}$  produced by  $E_{i-1}$  block. All the  $B_i$  blocks reconstruct the input at the scale of  $2^{-i}$ . Thus, all the outputs from  $B_i$  blocks create a spatial scale pyramid of the input image. Element-wise sigmoid as in (4.1) is used as activation function for  $B_i$  layers. Thus, different from works such as [166] and [167], we are not using the image pyramid as an input but as auxiliary training objective.



**FIGURE 4.10** Comparison between proposed 2D max-pooling-based label scaling and conventional label scaling. With conventional label scaling, the label is progressively scaled down using nearest neighbor interpolation (blue arrows  $\rightarrow$ ). In our proposal, the original label (bottom left) is first converted to one-hot vectors (top left) and then progressively scaled down by 2D max-pooling operation (red arrows  $\rightarrow$ ). The segmentation color codes in the label are described in Table 4.2. On the top row, classes rather than footwear, sunglasses, top and shorts are ignored.

### 4.3.2 Label Pooling

Previous works involving multiple-scale inputs or outputs only consider spatial scaling. In [166], they are used as an auxiliary training objective. In [167], they are used to create multi-scale fusion features. In [19] and [18], the network is trained with different spatial scaled versions of input and output to produce more robust features.

However, with spatial scaling, details from the original input eventually are lost at smaller scales. To avoid such a problem, instead of spatially scaling the label, we use max-pooling operation on one-hot label vectors to produce multiple scales of labels. As such, the existence of a segment is preserved even in the smallest scale. This process is illustrated in Figure 4.10.

In Figure 4.10, the spatial scaling operation makes the existence of segmentation vanished. After the first scale down operation, the segmentation of sunglasses has vanished. From the scale of  $4 \times 4$  to  $2 \times 2$ , the top segment has vanished. By the time of scaling down to  $1 \times 1$  pixel, all the segmentations vanished. On the other hand, the proposed label pool features retain all of the segmentation even at the smallest scale.

Shown in Figure 4.9,  $D_i$  blocks is guide-trained by the result of  $P_i$  blocks where  $i$  is the level number, and  $P_i$  blocks are 2D max-pooling operation on the input label. Their configuration is shown in Table 4.1. This is to avoid the detail loss when scaling down the label. Also shown in Table 4.1, the stride of  $P_i$  are matched with the stride of  $D_i$  and  $E_i$  blocks. Furthermore, the kernel size of  $P_i$  is also matched with the kernel size of  $D_i$ .

### 4.3.3 Training Objectives

With the segmentation prediction  $l_0$  coming from  $L_0$ , we utilize pixel-wise cross-entropy as a training objective.

$$H(t, l_0) = \frac{1}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{K-1} t_{ij} \log(l_{0ij}) \quad (4.3)$$

Where  $t$  is the ground-truth,  $t_{ij}$  is the  $j$ -th channel of the  $i$ -th pixel of  $t$ ,  $l_{0ij}$  is the  $j$ -th channel of the  $i$ -th pixel of  $l_0$ ,  $K$  is the number of segmentation class, and  $N$  is the total number of pixel in the output. Besides the conventional training criteria, we introduce the three auxiliary training objectives as follows.

#### 4.3.3.1 Image Pyramid Loss (IPL)

Different from popular works, we do not utilize multi-scaled input to reinforce the training process. Instead, we expect the network to reconstruct scaled versions of the input using feature maps produced by encoder blocks. Thus, additional scaled inputs and processing are not required during the inference process. We penalize the difference between output  $b_i$  from  $B_i$  block and the input image  $x_i$  by binary cross-entropy loss as follows.

$$H(x_i, b_i) = -\frac{1}{N} \sum_{j=0}^{N-1} \left( x_{ij} \cdot \log(b_{ij}) + (1 - x_{ij}) \cdot \log(1 - b_{ij}) \right) \quad (4.4)$$

Where  $x_i$  and  $b_i$  are input image and reconstructed image at scale  $2^{-i}$  with  $i \in [1..4]$ ,  $x_{ij}$  and  $b_{ij}$  are the  $j$ -th element of  $x_i$  and  $b_i$ ,  $N$  is the total number of elements in  $x_i$  and  $b_i$  (i.e. number of pixel  $\times$  number of color channels). We use binary cross-entropy (i.e., log loss) as an error function. Then, the image pyramid loss is calculated by:

$$IPL = \frac{1}{4} \sum_{i=1}^4 H(x_i, b_i) \quad (4.5)$$

#### 4.3.3.2 Segmentation Pyramid Loss (SPL)

It is the average of the cross-entropy between segmentation and ground truth across different scales.

$$SPL = \frac{1}{4} \sum_{i=1}^4 H(t_i, l_i) \quad (4.6)$$

Where  $H(\cdot)$  is binary cross-entropy loss similar to (4.4),  $t_i$  and  $l_i$  are ground truth and predicted segmentation at scale  $2^{-i}$  with  $i \in [1..4]$ .



### 4.3.3.3 Label Pooling Loss (LPL)

It is the average of binary cross-entropy loss between label pool features and output of decoders across different scales as follows.

$$LPL = \frac{1}{6} \sum_{i=1}^6 H(p_i, d_i) \quad (4.7)$$

Where  $H(\cdot)$  is binary cross-entropy loss similar to (4.4),  $p_i$  and  $d_i$  are ground truth and prediction of label pool feature at scale  $2^{-i}$ .

The final loss is calculated by taking average all the above-mentioned losses as follows:

$$loss = \frac{1}{4} (H(t, l_0) + IPL + SPL + LPL) \quad (4.8)$$

## 4.4 Setting and Evaluation

Using the ModaNet data set, we compare our model with U-Net [65], PSPNet [19], SegNet [17] and DeepLabv3+ [18].

















**FIGURE 4.11** Illustration of segmentation results. Photos are public domain works downloaded from Pexels.com. Label are manually annotated by the authors.

### 4.4.1 Data Set

We split the original training set into new training and evaluation sets. The new evaluation set consists of 2,400 images, and the new training set consists of the remaining 49,977 images.

TABLE 4.2 ModaNet Data Set Statistic

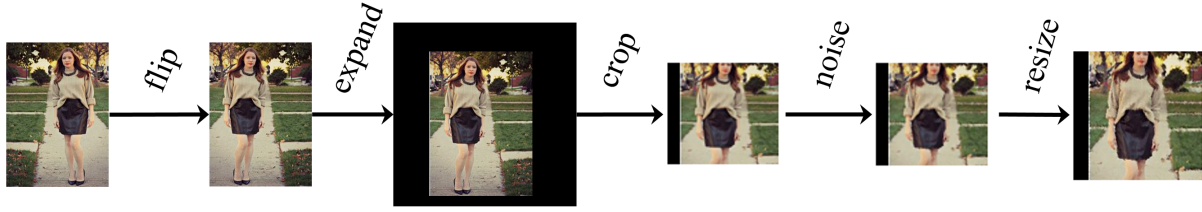
Id.	Color	Class	Inst. Count		Avg Inst. Size	
			Train	Val	Train	Val
0		Background	–	–	–	–
1		Bag	19,603	948	2.46%	2.53%
2		Belt	13,081	636	0.46%	0.44%
3		Boots	6,719	365	2.40%	2.36%
4		Footwear	37,468	1,753	0.94%	0.93%
5		Outer	22,597	1,093	7.43%	7.42%
6		Dress	13,764	662	10.46%	10.52%
7		Sunglasses	8,340	411	0.30%	0.30%
8		Pants	21,950	1,064	5.65%	5.47%
9		Top	33,131	1,544	4.79%	5.04%
10		Shorts	6,709	322	2.75%	2.83%
11		Skirts	12,953	622	6.37%	6.23%
12		Headwear	5,164	281	1.22%	1.21%
13		Scarf & Tie	4,711	284	2.85%	3.20%

The randomized splitting process is constrained so that there are at least 280 instances of each class available in the evaluation set to ensure the quality of evaluation. The statistic of training and validation data sets are shown in Table 4.2.

### 4.4.2 Data Augmentation

We train and evaluate all the networks with input and output sizes of  $224 \times 224$ . To make the networks more robust, the following pipeline is used for data augmentation:

- 1) Random horizontal flipping
- 2) Random expanding with a max expansion ratio of 1.5. In this step, black bars of random size  $t$ ,  $b$ ,  $l$  and  $r$  are padded into the original image so that  $l + r \leq 0.5 \times w$  and  $t + b \leq 0.5 \times h$ , where  $w$  and  $h$  are width and height of the input of this step,  $t$  and  $b$  are the sizes of black bars padded on the top and bottom of the image, and  $l$  and  $r$  are the sizes of black bars padded on the left and right side of the image.



**FIGURE 4.12** Illustration of image augmentation used for the training process. Photos are public domain works downloaded from Pexels.com.

3) Randomly cropping the image with scale ratio range  $(0.5, 1]$  and aspect ratio range  $[\frac{3}{4}, \frac{4}{3}]$ . Thus, width  $w$  and height  $h$  of the cropping window are randomized so that:

- $w \leq w_0$  and  $h \leq h_0$
- $\frac{3}{4} \leq \frac{w}{h} \leq \frac{4}{3}$
- $0.5 \times (w_0 \times h_0) < w \times h \leq w_0 \times h_0$

Where  $w_0$  and  $h_0$  are the width and height of the input of this step. Hence, the top left corner of the cropping window  $(x, y)$  must satisfy the following conditions:

- $0 \leq x \leq w_0 - w$
- $0 \leq y \leq h_0 - h$

4) Adding Gaussian noise with mean  $\mu = 0$  and standard deviation  $\sigma = 25.5$ . Thus, the output image is  $x = \min(255, \max(0, x + G(\mu, \sigma)))$  where  $G(\cdot)$  is the Gaussian function.

5) Resize to  $224 \times 224 \times 3$

Figure 4.12 illustrates the augmentation process. As illustrated, the output images could have some details cropped away. However, with the augmentation parameter as described above, we find that majority of the subject in the ModaNet data set is preserved.

We then normalize the input image by scaling pixel value into  $[0, 1]$  range. Since the label is an image containing pixel-level segmentation of the input image, it also needs to be augmented correspondingly, except for the step 4. Furthermore, nearest-neighbor sampling must be used in all the steps that involve interpolation to preserve class information.

### 4.4.3 Metrics

#### 4.4.3.1 mean Intersection over Union (mIoU)

We utilize Intersection over Union (IoU), i.e., Jaccard distance as the performance metric. We first compute the IoU of individual class as follows.

$$IoU_i = \frac{1}{N} \sum_{j=0}^{N-1} \frac{|T_{ij} \cap L_{ij}|}{|T_{ij} \cup L_{ij}|} \quad (4.9)$$

Where  $IoU_i$  is the IoU score of class  $i$ ,  $N$  is the total number of photos in the data set,  $T_{ij}$  is the set of all the pixels belongs to the  $i$ -th class in  $j$ -th ground truth,  $L_{ij}$  is the set of all pixels predicted as  $i$ -th class in the  $j$ -th prediction, and  $|\cdot|$  is the cardinality of a set. Thus, the mIoU metric is calculated as follows.

$$mIoU = \frac{1}{M} \sum_{i=0}^{M-1} IoU_i \quad (4.10)$$

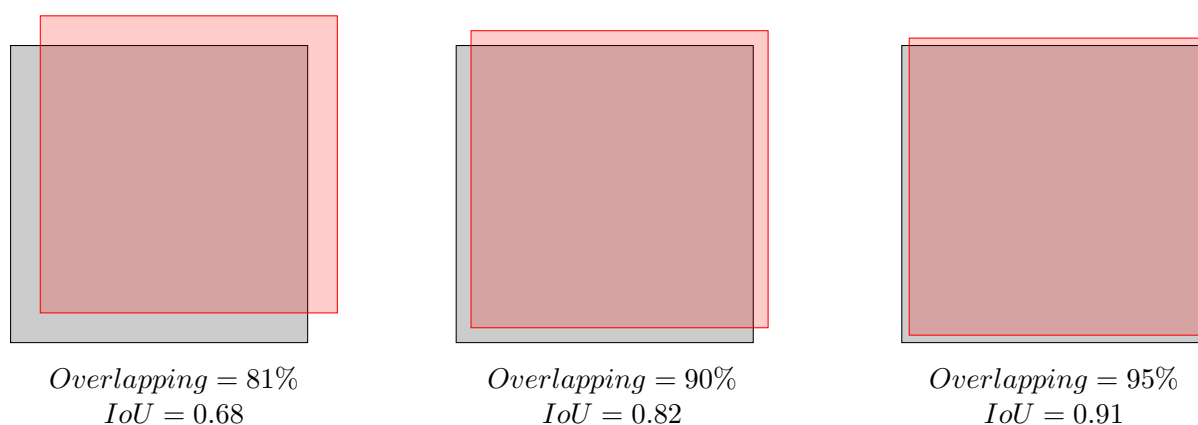


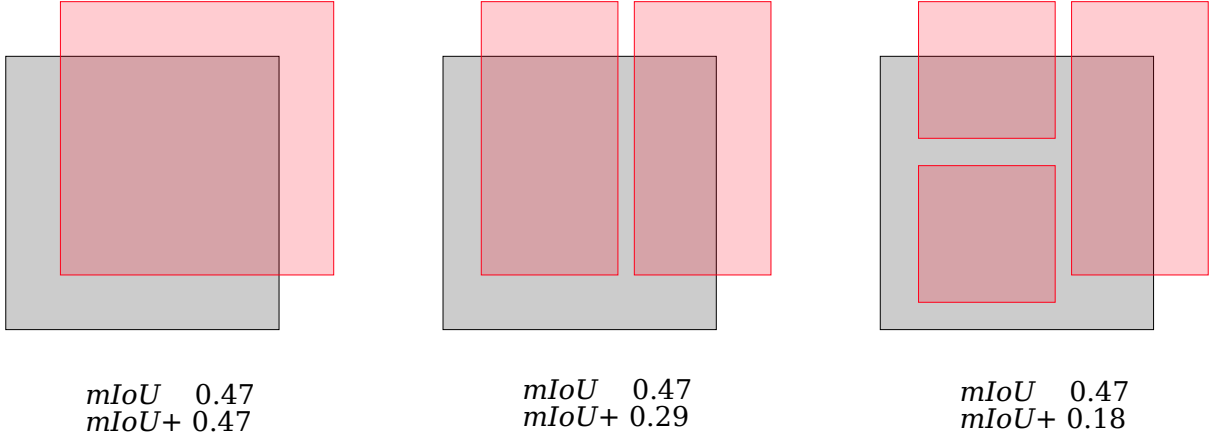
FIGURE 4.13 Illustration of IoU Metric

Where  $M$  is the total number of segmentation classes. Figure 4.13 illustrates the IoU score in three different cases. The score is calculated by dividing the overlapping area by the union area between the gray square and the red square. To add more clarity, the union area is calculated by the sum of both squares minus the overlapping area. As shown in the figure, the IoU metric is quite severe compared to the intersection percentage metric.

#### 4.4.3.2 mean Intersection over Union Plus (mIoU+)

Because the mIoU metric favors the total number of accurately classified pixels, a prediction with noise frequently results in higher mIoU compared to a prediction with no noise. Depending on the application, prediction with low noise may be favored over absolutely high mIoU

prediction.



**FIGURE 4.14** Comparison between mIoU and mIoU+ metric.

Therefore, we propose Intersection over Union Plus (IoU+) metric in which noise is taken into account. This metric is not based on individual pixels but connected components (i.e., individual segments). Figure 4.14 shows the comparison between mIoU and mIoU+ metrics. As shown in the figure, although all the samples have the same mIoU score, their mIoU+ scores are different.

Given a prediction and a ground truth segmentation, the connected component based segmentation score of a class is calculated as follows.

$$CCSS_i(U, V) = \frac{1}{|U_i|} \sum_{u \in U_i} \max_{v \in V_i} IoU(u, v) \quad (4.11)$$

Where  $CCSS_i$  is segmentation score of the  $i$ -th class between predicted segmentation  $U$  and ground truth  $V$ ,  $U_i$  is the set of all connected components of  $i$ -th class in the prediction,  $V_i$  is the set of all connected components of  $i$ -th class in the ground truth. However, because this score is not symmetric, i.e.  $CCSS_i(U, V) \neq CCSS_i(V, U)$ , the segmentation score is calculated as follows.

$$SS_i(U, V) = CCSS_i(U, V) \wedge CCSS_i(V, U) \quad (4.12)$$

Where  $SS_i(U, V)$  is the segmentation score of the  $i$ -th class between two segmentations  $U$  and  $V$ . Similar to the conventional IoU, IoU+ of each class is computed as follow.

$$IoU+_i = \frac{1}{N} \sum_{j=0}^{N-1} SS_i(U_j, V_j) \quad (4.13)$$

Where  $IoU+_i$  is IoU+ score of the  $i$ -th class,  $N$  is the total number of samples,  $U_j$  is the set of connected components from the  $j$ -th prediction, and  $V_j$  is the set of connected components from

the  $j$ -th ground truth. The score for a whole segmentation with multiple connected components is calculated as follows.

$$mIoU+ = \frac{1}{K} \sum_{i=0}^{K-1} IoU+_i \quad (4.14)$$

Where  $K$  is the total number of segmentation classes. Figure 4.15 illustrates the IoU+ metric. All the coordinates of all the bounding boxes are provided.  $v_1$  and  $v_2$  are two connected components of segmentation set  $V$ .  $u_1$  is the only one member of segmentation set  $U$ .

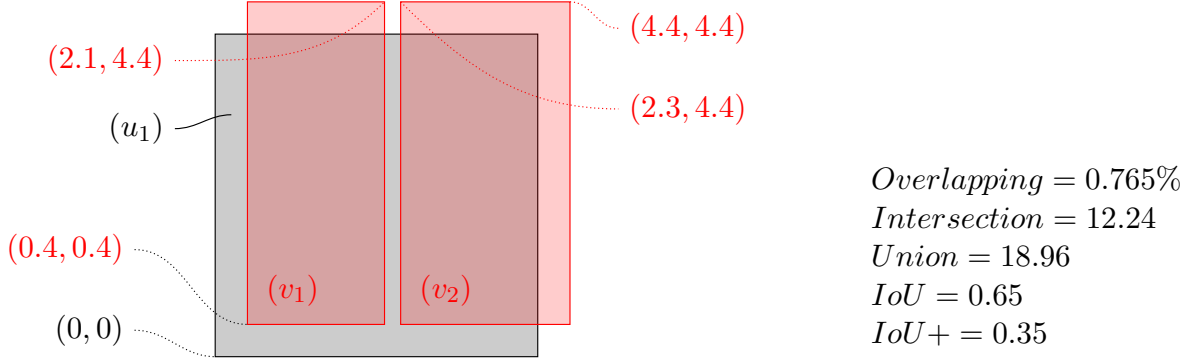


FIGURE 4.15 Illustration of IoU+ Metric

As shown in the figure, the IoU score is 0.65 and can be considered as good. However, the IoU+ score is only 0.35, significantly lower than the conventional IoU metric. It is due to the fragmentation of the red area. To calculate the IoU+ score, following IoU scores need to be calculated.

$$IoU(u_1, v_1) = IoU(v_1, u_1) = \frac{u_1 \cap v_1}{u_1 \cup v_1} = \frac{6.12}{16.68} = 0.37$$

$$IoU(u_1, v_2) = IoU(v_2, u_1) = \frac{u_1 \cap v_2}{u_1 \cup v_2} = \frac{6.12}{18.28} = 0.33$$

Thus, the connected component segmentation scores are calculated as follows.

$$\begin{aligned} CCSS(U, V) &= \frac{1}{|U|} \sum_{u \in U} \max_{v \in V} IoU(u, v) \\ &= \frac{1}{1} \max_{v \in V} IoU(u_1, v) \\ &= \max \{IoU(u_1, v_1), IoU(u_1, v_2)\} \\ &= \max \{0.37, 0.33\} \\ &= 0.37 \end{aligned}$$

$$\begin{aligned}
CCSS(V, U) &= \frac{1}{|V|} \sum_{v \in V} \max_{u \in U} IoU(v, U) \\
&= \frac{1}{2} \left( \max_{u \in U} IoU(u, v_1) + \max_{u \in U} IoU(u, v_2) \right) \\
&= 0.5 (\max \{IoU(u_1, v_1)\} + \max \{IoU(u_1, v_2)\}) \\
&= 0.5 \times (IoU(u_1, v_1) + IoU(u_1, v_2)) \\
&= 0.35
\end{aligned}$$

Finally, mIoU+ is calculated as follows:

$$IoU+(U, V) = CCSS(U, V) \wedge CCSS(V, U) = 0.35$$

#### 4.4.4 Ablation Study on Effect of Auxiliary Training Objectives

We investigate more into the effect of auxiliary training objective on the model performance. We retrain our network with different training objective configurations. The loss function used in this experiment is as follows.

$$loss = \frac{H(t_0, l_0) + \alpha IPL + \beta SPL + \gamma LPL}{1 + \alpha + \beta + \gamma} \quad (4.15)$$

Where  $\alpha, \beta, \gamma \in \{0, 1\}$ . In practice, when  $\alpha = 0$ ,  $b_i$  computations are ignored. Similarly, when  $\beta = 0$ ,  $l_{[1..4]}$  computations are ignored. However, when  $\gamma = 0$ ,  $d_i$  still need to be computed because it is an integrated part of the model.

**TABLE 4.3** Different Auxiliary Configurations

	A	B	C	D	E	F	G	H
$\alpha$	1	1	1	1	0	0	0	0
$\beta$	1	1	0	0	1	1	0	0
$\gamma$	1	0	1	0	1	0	1	0

There are 8 different configurations of the loss function. We annotate them as configuration A to configuration H, as shown in Table 4.3.

#### 4.4.5 Settings

We implement all the networks using Chainer deep learning framework [168]. LeCun normal weight initializer [169] is used. The models are trained by an improved version of Adam

optimizer [105] called AdamW [170]. For reference, Algorithm 5 describes the algorithm of both Adam and AdamW. Where  $t$  is the training time step,  $\mathbf{x}$  is the parameter vector,  $\mathbf{m}$  is the first-moment vector,  $\mathbf{v}$  is the second-moment vector, and  $\eta$  is the schedule multiplier. The pink and green highlights indicate the parts that only available in Adam and AdamW, respectively.

---

**Algorithm 5** Adam and AdamW

---

```

1: given  $\alpha_t = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$ 
2: initialize  $t \leftarrow 0, \mathbf{m}_{t=0} \leftarrow 0, \mathbf{v}_{t=0} \leftarrow 0, \mathbf{x}_{t=0} \in \mathbb{R}^n, \eta_{t=0} \in \mathbb{R}$ 
3: repeat
4:    $t \leftarrow t + 1$ 
5:    $\nabla f_t(\mathbf{x}_{t-1}) \leftarrow \text{SelectBatch}(\mathbf{x}_{t-1})$  ▷ select batch and return the gradient
6:    $\mathbf{g}_t \leftarrow \nabla f_t(\mathbf{x}_{t-1}) + w_t \mathbf{x}_{t-1}$ 
7:    $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$ 
8:    $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$ 
9:    $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$  ▷  $\beta_1$  is taken to the power of  $t$ 
10:   $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$  ▷  $\beta_2$  is taken to the power of  $t$ 
11:   $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ 
12:   $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \eta_t \left( \alpha_t \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon) + w_t \mathbf{x}_{t-1} \right)$ 
13: until stopping criterion is met
14: return optimized parameters  $\mathbf{x}_t$ 

```

---

In our implementation, all the parameters are set to default values, as in Algorithm 5. However, the weight decay rate  $w$  is set to 0.99. This weight decay rate is constant throughout the training process; i.e.,  $w_t$  is always equal to 0.99. Furthermore,  $\eta_t$  always equals to 1.

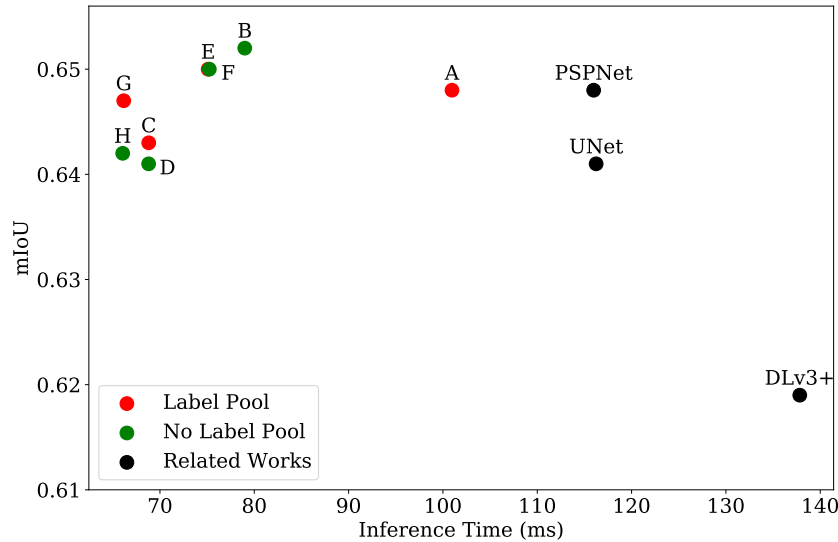
The machine used to carry out the experiment is a Linux box equipped with three Nvidia Pascal GPUs. We train each configuration for three times with 100 epochs each and take averages of the best mIoU and mIoU+.

#### 4.4.6 Result

Figure 4.16 compares the networks' performance under mIoU metric. The detailed experiment result is shown in Table 4.4 and Figure 4.16. Our proposed network configured with three different auxiliary losses, outperforms all the ever-proposed models in terms of performance. Among all the auxiliary loss configurations, configuration B achieves the highest mIoU score. This configuration consists of only image pyramid loss and segmentation pyramid loss. Among the ever-proposed networks, PSPNet achieves the highest mIoU score. Moreover, our top proposed network takes only 2/3 of the time for training as well as inference compared to PSPNet.

As shown in Figure 4.16, the performance of the model is worsened when combining label pooling loss with the other two auxiliary training losses. In fact, configuration B, D, and F achieve higher mIoU compared to configuration A, C, and E.





**FIGURE 4.16** Network Performance under mIoU metric

Figure 4.17 compares the networks’ performance using mIoU+ metric. The detailed experimental results are shown in Table 4.5. The training and inference time in Table 4.5 are carried over from Table 4.4. We observe that all the models achieve their best mIoU and mIoU+ in the same epoch. Furthermore, mIoU and mIoU+ are loosely proportional to each other during the training process.

As shown in Figure 4.17, the proposed model with configuration E achieves the highest mIoU+ score. Configuration E consists of segmentation pyramid loss and label pooling loss. Configuration A with all the auxiliary loss functions is the runner-up. Among the ever-proposed model, PSPNet also achieves the highest mIoU+ score.

The proposed model used to generate samples in Figure 4.11 and Figure 4.18 is trained with configuration A.

TABLE 4.4 IoU of Individual Classes

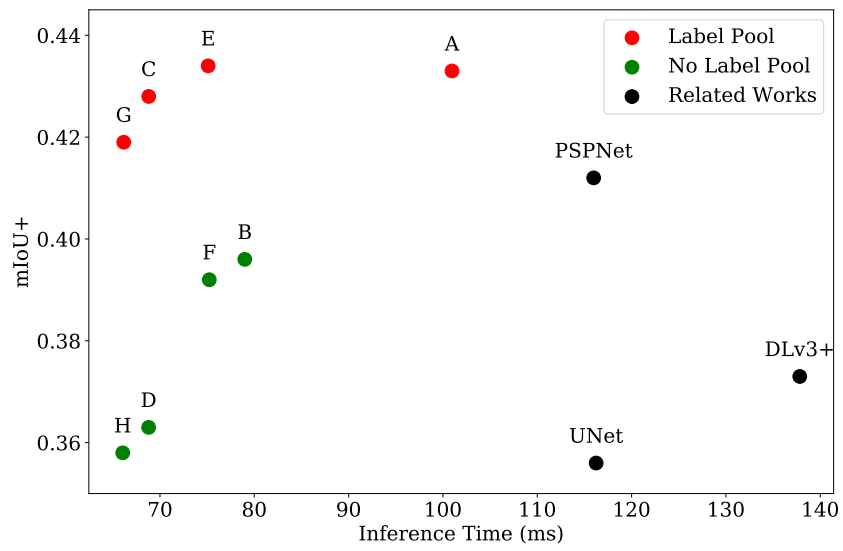
	A	B	C	D	E	F	G	H	Unet	DLv3+ <sup>a</sup>	PSPNet	SegNet
Background	0.979	<u>0.979</u>	0.978	0.979	0.979	<b>0.980</b>	0.978	0.979	0.979	0.975	0.977	0.955
Bag	0.690	0.692	0.691	0.694	0.693	0.689	<u>0.707</u>	0.694	<i>0.701</i>	0.674	<b>0.708</b>	0.429
Belt	0.465	0.462	<u>0.467</u>	0.442	0.456	0.454	0.454	0.440	<b>0.479</b>	0.394	0.415	0.205
Boots	0.556	<b>0.577</b>	<u>0.570</u>	0.543	<u>0.575</u>	0.557	0.569	0.559	0.561	0.535	0.567	0.369
Footwear	0.630	0.624	0.629	0.628	<b>0.638</b>	<u>0.638</u>	0.631	0.622	0.637	0.586	0.555	0.452
Outer	0.642	0.644	0.623	0.629	0.639	<u>0.651</u>	0.627	0.638	0.623	0.625	<b>0.665</b>	0.438
Dress	<u>0.669</u>	0.668	0.633	0.663	0.651	0.657	0.651	0.649	0.594	0.660	<b>0.689</b>	0.462
Sunglasses	0.634	0.611	<u>0.652</u>	0.606	0.650	0.625	0.646	0.621	<b>0.675</b>	0.531	0.534	0.321
Pants	0.805	<b>0.810</b>	0.793	0.790	0.802	<u>0.808</u>	0.801	0.792	0.787	0.761	0.800	0.683
Top	0.647	0.650	0.625	0.641	0.653	<u>0.660</u>	0.641	0.652	0.624	0.610	<b>0.679</b>	0.478
Shorts	0.686	<b>0.718</b>	0.686	0.697	0.697	0.692	0.688	0.708	0.662	<u>0.715</u>	0.711	0.487
Skirts	0.661	0.674	0.659	0.671	0.646	0.673	0.666	0.652	0.618	<u>0.683</u>	<b>0.709</b>	0.503
Headwear	<u>0.608</u>	0.586	0.606	0.584	0.582	0.590	0.606	0.590	<b>0.618</b>	0.545	0.594	0.258
Scarf & Tie	0.393	0.429	0.396	0.409	<u>0.439</u>	0.426	0.397	0.395	0.420	0.370	<b>0.473</b>	0.129
mIoU	0.648	<b>0.652</b>	0.643	0.641	0.650	<u>0.650</u>	0.647	0.642	0.641	0.619	0.648	0.441
inference time (ms)	100.96	78.99	68.81	68.80	75.11	75.23	66.16	<u>66.05</u>	116.24	137.83	115.98	<b>29.61</b>
training time (h)	19.568	19.328	17.825	17.625	19.359	19.297	17.831	<u>17.483</u>	30.84	33.64	23.68	<b>9.04</b>

<sup>a</sup>DeepLabv3+

TABLE 4.5 IoU+ of Individual Classes

	A	B	C	D	E	F	G	H	Unet	DLv3+ <sup>a</sup>	PSPNet	SegNet
Background	0.373	0.344	0.369	0.336	0.359	0.347	0.339	0.338	0.345	0.379	<b>0.404</b>	0.233
Bag	<b>0.441</b>	0.398	0.434	0.361	0.429	0.392	0.424	0.385	0.373	0.364	0.427	0.124
Belt	0.290	0.257	0.312	0.245	<b>0.316</b>	0.257	0.297	0.240	0.287	0.204	0.211	0.082
Boots	0.347	0.303	0.358	0.269	0.335	0.297	<b>0.360</b>	0.280	0.297	0.299	0.301	0.100
Footwear	0.502	0.474	0.503	0.483	0.503	0.489	<b>0.504</b>	0.475	0.500	0.434	0.418	0.247
Outer	0.412	0.379	0.383	0.356	0.409	0.375	0.408	0.329	0.286	0.381	<b>0.425</b>	0.104
Dress	0.425	0.393	0.370	0.336	0.398	0.382	0.393	0.360	0.208	0.366	<b>0.434</b>	0.082
Sunglasses	0.501	0.471	<b>0.541</b>	0.433	0.531	0.463	0.514	0.402	0.502	0.355	0.392	0.157
Pants	0.629	0.613	<b>0.642</b>	0.568	0.635	0.621	0.623	0.567	0.557	0.591	0.641	0.316
Top	0.432	0.439	0.438	0.394	0.452	0.428	0.401	0.391	0.347	0.406	<b>0.465</b>	0.173
Shorts	0.488	0.426	0.490	0.393	<b>0.527</b>	0.454	0.449	0.374	0.364	0.453	0.497	0.203
Skirts	0.459	0.413	0.438	0.370	0.465	0.426	0.454	0.390	0.297	0.436	<b>0.497</b>	0.163
Headwear	<b>0.479</b>	0.399	0.468	0.362	0.452	0.379	0.467	0.310	0.441	0.336	0.400	0.083
Scarf & Tie	<b>0.278</b>	0.230	0.239	0.185	0.270	0.186	0.238	0.172	0.184	0.211	0.263	0.049
mIoU+	0.433	0.396	0.428	0.363	<b>0.434</b>	0.392	0.419	0.358	0.356	0.373	0.412	0.151
inference time (ms)	100.96	78.99	68.81	68.80	75.11	75.23	66.16	66.05	116.24	137.83	115.98	<b>29.61</b>
training time (h)	19.568	19.328	17.825	17.625	19.359	19.297	17.831	17.483	30.84	33.64	23.68	<b>9.04</b>

<sup>a</sup>DeepLabv3+



**FIGURE 4.17** Network Performance under mIoU+ metric



**FIGURE 4.18** Illustration of segmentation results coming from the models. Photos are public domain works downloaded from Pexels.com. Label are manually annotated by the authors.



## CHAPTER 5

# Conclusion

This thesis described our two works on deep learning (DL): Scalable Vector Graphic AI (SvgAI) and Street Fashion Semantic Segmentation (SFSS). Both of the works involved in extracting semantic meta from visual information. Thus, they are both aligned to the general trend of DL research that mentioned in [Chapter 1](#). Moreover, the high degree of the practicality of the works positioned themselves into our zone of research interest. Though SvgAI and SFSS, we showed that artificial intelligence (AI) could penetrate more into daily life with an appropriate strategy.

## 5.1 SvgAI

In SvgAI, we introduced an intelligent agent (IA) that can draw semantic Scalable Vector Graphics (SVG) image. We proposed a framework to train the agent to use SVG editor by using Q-learning and policy-gradient. To successfully train the agent by using Q-learning, we divide the action space into two sets and apply independent exploration policies on each action set. Evaluation results show the efficiency of the proposed dual  $\epsilon$ -greedy policy and policy-gradient. The detail evaluation of SvgAI is described in [Section 3.5.2](#). The SVG image produced by the proposed agent not only retains the semantic structure but also higher in visual quality compared to the conventional image processing methods.

Ultimately, a semantical SVG is not only about using the correct shape and fine gradient. A completed SVG might only contains paths elements. However, SVG created by humans are mostly semantically organized by using group elements. For example, all the paths elements that constitute the wheel on an image of a car should go to a group named *wheel*.

By organizing the image elements semantically, it is easier for multiple artists to collaborate on the same project. This semantic grouping not only valid in composing vector images but also in the raster images. Popular raster image editing software mostly allows the user to divide an image into layers, and within a layer, it is possible to have a hierarchical grouping scheme. Even in works like programming, programmers likewise need to divide a program into different

modules, class corresponding to their semantic function.

Thus, having an ability to organize an SVG document semantically is essential for human and AI collaboration. Therefore, semantic SVG in group level would be a longer term target for SvgAI. In this level, besides synthetic data, the AI can also be trained using publicly available SVG files.

## 5.2 SFSS

In SFSS, we propose a high-performance semantic segmentation model for street fashion photos. We also propose a new label pool feature that greatly performance of the proposed network. For better evaluation, we propose the mIoU+ metric in which noises are taken into account. The experiment shows that our network requires less time to train and infer while achieves the highest segmentation performance in both mean Intersection over Union (mIoU) and mIoU+ metrics. The detailed evaluation of this work is described in [Section 4.4.6](#).

With label pooling training objective, the network outputs feature pooling features at different scales in the decoder side. Thus, the network infers the existence of the classes over the whole image. Then, through multiple layers of decoders, it progressively refines the result up to the desired resolution. However, in the case of a large number of classes, the current design of label pooling requires the size of the decoder to be increased. Embedding this feature into a smaller feature space seems to be a linear solution. This leads to a more general view of the label pool training scheme. In this view, the network is guided to learn the described features. Hence, it narrows down the descent path of the network and leads to the desired property.

## 5.3 Related Issues

The direction of AI development in the last decade was depicted by the availability of datasets and toolboxes. It was the ImageNet [7] that leads to renewed interest in convolutional neural network (CNN). Similarly, OpenAI Gym [155] that leads to a series of AI milestones after 2015. Other notable datasets are CityScape [67], COCO [66], LFW [171].

These datasets and toolboxes waived the burden of collecting and distilling raw data from the research community. Therefore, it boosts the research output as a whole. However, the number of datasets and toolboxes are still limited. Creating a dataset is tedious and expensive. Therefore the lack of dataset prevents an individual and small organization from pursuing a specific research target.

There are efforts in developing AI that learn with fewer data such as Omniglot Challenge [172]. However, this direction is not received much attention it deserved compared to



other trending topics such as natural language processing (NLP) [173, 174]. More importantly, even a small dataset is still expensive. While data is out of reach for individual and small organization, large corporation such as Google, Facebook, Amazon own massive amount of data. This leads to a new concern in AI fairness and privacy. According to [6], the total number of AI mentions in the legislation of the 2017–2018 US congress in increased tenfold compared to the previous congress. This trend is expected to continue with the 2019–2020 congress.

Thus, to make AI more accessible to the mass, more progress need to be done in multiple fronts: (1) better legislation to combat privacy and fairness concern, (2) more datasets and tool-boxes, (3) better AI model and implementation strategy. Out of the three fronts, SvgAI, and SFSS offers a better model and implementation strategy and achieved state-of-the-art performance in their domain.



# Bibliography

- [1] D. Poole, A. Mackworth, and R. Goebel, “Computational intelligence: a logical approach. 1998,” *Google Scholar Google Scholar Digital Library Digital Library*, 1998.
- [2] S. Russell and P. Norvig, “Artificial intelligence: a modern approach,” 2002.
- [3] N. J. Nilsson and N. J. Nilsson, *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.
- [4] S. Legg, M. Hutter *et al.*, “A collection of definitions of intelligence,” *Frontiers in Artificial Intelligence and applications*, vol. 157, p. 17, 2007.
- [5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” pp. 1097–1105, 2012.
- [6] R. Perrault, Y. Shoham, E. Brynjolfsson, J. Clark, J. Etchemendy, B. Grosz, T. Lyons, J. Manyika, S. Mishra, and J. C. Niebles, “The ai index 2019 annual report,” *AI Index Steering Committee, Human-Centered AI Institute, Stanford University, Stanford, CA*, 2019.
- [7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [8] A. H. Dang and W. Kameyama, “Svgai—training methods analysis of artificial intelligent agent to use svg editor,” *ICTACT Transactions on Advanced Communications Technology (TACT)*, vol. 7, no. 6, pp. 1159–1166, 2018.
- [9] T. Beltramelli, “pix2code: Generating code from a graphical user interface screenshot,” in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2018, pp. 1–6.

- [10] A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.
- [11] A. H. Dang and W. Kameyama, “Robust semantic segmentation for street fashion photos,” *ICACT Transactions on Advanced Communications Technology (TACT)*, vol. 8, no. 6, pp. 1248–1257, 2019.
- [12] S. Zheng, F. Yang, M. H. Kiapour, and R. Piramuthu, “Modanet: A large-scale street fashion dataset with polygon annotations,” *arXiv preprint arXiv:1807.01394*, 2018.
- [13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE, jun 2015, pp. 1–9, inceptionv1.
- [14] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning*, 2015, pp. 448–456, inception-v2.
- [15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.
- [16] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI’17. AAAI Press, 2017, p. 4278–4284, inception-v4.
- [17] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.
- [18] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 801–818.
- [19] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

- [20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119.
- [21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
- [22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [23] L. A. Gatys, A. S. Ecker, and M. Bethge, “A neural algorithm of artistic style,” *arXiv preprint arXiv:1508.06576*, 2015.
- [24] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, “Photo-realistic single image super-resolution using a generative adversarial network,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.
- [25] J. Johnson, A. Alahi, and L. Fei-Fei, “Perceptual losses for real-time style transfer and super-resolution,” in *European conference on computer vision*. Springer, 2016, pp. 694–711.
- [26] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, “Autoencoding beyond pixels using a learned similarity metric,” *arXiv preprint arXiv:1512.09300*, 2015.
- [27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.
- [28] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.
- [29] M. I. Jordan, “Attractor dynamics and parallelism in a connectionist sequential machine,” in *Artificial neural networks: concept learning*, 1990, pp. 112–127.
- [30] B. A. Pearlmutter, “Learning state space trajectories in recurrent neural networks,” *Neural Computation*, vol. 1, no. 2, pp. 263–269, 1989.
- [31] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, “Finite state automata and simple recurrent networks,” *Neural computation*, vol. 1, no. 3, pp. 372–381, 1989.

- [32] M. H. Segler, T. Kogej, C. Tyrchan, and M. P. Waller, “Generating focused molecule libraries for drug discovery with recurrent neural networks,” *ACS central science*, vol. 4, no. 1, pp. 120–131, 2018.
- [33] A. Vetrivel, M. Gerke, N. Kerle, F. Nex, and G. Vosselman, “Disaster damage detection through synergistic use of deep learning and 3d point cloud features derived from very high resolution oblique aerial images, and multiple-kernel-learning,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 140, pp. 45 – 59, 2018, geospatial Computer Vision. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0924271616305913>
- [34] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3104–3112. [Online]. Available: <http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf>
- [35] K.-F. Lee and S. Mahajan, “Bill: a table-based, knowledge-intensive othello program,” 1986.
- [36] J. Schaeffer, R. Lake, P. Lu, and M. Bryant, “Chinook the world man-machine checkers champion,” *AI Magazine*, vol. 17, no. 1, pp. 21–21, 1996.
- [37] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, “Deep blue,” *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.
- [38] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*, 2016, pp. 1928–1937.
- [39] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, p. 484, 2016.
- [40] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, “Dermatologist-level classification of skin cancer with deep neural networks,” *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.
- [41] W. Xiong, L. Wu, F. Allewa, J. Droppo, X. Huang, and A. Stolcke, “The microsoft 2017 conversational speech recognition system,” in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018, pp. 5934–5938.

- [42] G. Saon, G. Kurata, T. Sercu, K. Audhkhasi, S. Thomas, D. Dimitriadis, X. Cui, B. Ramabhadran, M. Picheny, L.-L. Lim *et al.*, “English conversational telephone speech recognition by humans and machines,” *arXiv preprint arXiv:1703.02136*, 2017.
- [43] N. Brown, T. Sandholm, and S. Machine, “Libratus: The superhuman ai for no-limit poker.” in *IJCAI*, 2017, pp. 5226–5228.
- [44] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, “Deepstack: Expert-level artificial intelligence in heads-up no-limit poker,” *Science*, vol. 356, no. 6337, pp. 508–513, 2017.
- [45] H. Hassan, A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li *et al.*, “Achieving human parity on automatic chinese to english news translation,” *arXiv preprint arXiv:1803.05567*, 2018.
- [46] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell *et al.*, “Alphastar: Mastering the real-time strategy game starcraft ii,” *DeepMind blog*, p. 2, 2019.
- [47] Y. Lecun, *Generalization and network design strategies*. Elsevier, 1989.
- [48] H. S. Al-Khaffaf, A. Z. Talib, and R. A. Salam, “Empirical performance evaluation of raster-to-vector conversion methods: A study on multi-level interactions between different factors,” *IEICE TRANSACTIONS on Information and Systems*, vol. 94, no. 6, pp. 1278–1288, 2011.
- [49] V. Lacroix, “Raster-to-vector conversion: problems and tools towards a solution a map segmentation application,” in *2009 Seventh International Conference on Advances in Pattern Recognition*. IEEE, 2009, pp. 318–321.
- [50] R. Kansal and S. Kumar, “A framework for detection of linear gradient filled regions and their reconstruction for vector graphics,” 2013.
- [51] K. Kawamura, H. Watanabe, and H. Tominaga, “Vector representation of binary images containing halftone dots,” in *2004 IEEE International Conference on Multimedia and Expo (ICME)(IEEE Cat. No. 04TH8763)*, vol. 1. IEEE, 2004, pp. 335–338.
- [52] P. Selinger, “Potrace: a polygon-based tracing algorithm,” *Potrace (online)*, <http://potrace.sourceforge.net/potrace.pdf> (2009-07-01), 2003.
- [53] X. Hilaire and K. Tombre, “Robust and accurate vectorization of line drawings,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 6, pp. 890–904, 2006.

- [54] “Extensible markup language (xml) 1.0 (fifth edition),” Oct 2018, [Online; accessed 7. Feb. 2020]. [Online]. Available: <https://www.w3.org/TR/REC-xml>
- [55] S. Hochreiter and J. u. r. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [56] J. Shotton, M. Johnson, and R. Cipolla, “Semantic texton forests for image categorization and segmentation,” in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.
- [57] J. Shotton, A. W. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, “Real-time human pose recognition in parts from single depth images.” in *Cvpr*, vol. 2, 2011, p. 3.
- [58] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.
- [59] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” *arXiv preprint arXiv:1511.07122*, 2015.
- [60] G. Lin, A. Milan, C. Shen, and I. Reid, “Refinenet: Multi-path refinement networks for high-resolution semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1925–1934.
- [61] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Semantic image segmentation with deep convolutional nets and fully connected crfs,” *arXiv preprint arXiv:1412.7062*, 2014.
- [62] ———, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.
- [63] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *International Conference on Learning Representations*, 2015.
- [64] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [65] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.



- [66] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *European conference on computer vision*. Springer, 2014, pp. 740–755.
- [67] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The cityscapes dataset for semantic urban scene understanding,” in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [68] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, “Scene parsing through ade20k dataset,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [69] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vision*, vol. 60, no. 2, p. 91–110, Nov. 2004.
- [70] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (surf),” *Comput. Vis. Image Underst.*, vol. 110, no. 3, p. 346–359, Jun. 2008. [Online]. Available: <https://doi.org/10.1016/j.cviu.2007.09.014>
- [71] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *Proceedings of the 2011 International Conference on Computer Vision*, ser. ICCV ’11. USA: IEEE Computer Society, 2011, p. 2564–2571. [Online]. Available: <https://doi.org/10.1109/ICCV.2011.6126544>
- [72] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, p. 273–297, Sep. 1995. [Online]. Available: <https://doi.org/10.1023/A:1022627411411>
- [73] T. K. Ho, “Random decision forests,” in *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ser. ICDAR ’95. USA: IEEE Computer Society, 1995, p. 278.
- [74] —, “The random subspace method for constructing decision forests,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, p. 832–844, Aug. 1998. [Online]. Available: <https://doi.org/10.1109/34.709601>
- [75] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
- [76] D. O. Hebb and D. Hebb, *The organization of behavior*. Wiley New York, 1949, vol. 65.

- [77] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and adaptive systems: fundamentals through simulations*. Wiley New York, 2000, vol. 672.
- [78] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [79] P. Werbos, “Beyond regression:” new tools for prediction and analysis in the behavioral sciences,” *Ph. D. dissertation, Harvard University*, 1974.
- [80] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.
- [81] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.
- [82] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines,” *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.
- [83] P. Smolensky, “Information processing in dynamical systems: Foundations of harmony theory,” Colorado Univ at Boulder Dept of Computer Science, Tech. Rep., 1986.
- [84] Y. LeCun, L. e. o. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [85] E. Oja, “Simplified neuron model as a principal component analyzer,” *Journal of mathematical biology*, vol. 15, no. 3, pp. 267–273, 1982.
- [86] T. D. Sanger, “Optimal unsupervised learning in a single-layer linear feedforward neural network,” *Neural networks*, vol. 2, no. 6, pp. 459–473, 1989.
- [87] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators.” *Neural Networks*, 1989.
- [88] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [89] A. Karpathy. (2014) Hacker’s guide to neural networks. [Online]. Available: <http://karpathy.github.io/neuralnets/>
- [90] A. Cauchy, “Méthode générale pour la résolution des systemes d’équations simultanées,” *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.

- [91] H. Robbins and S. Monro, “A stochastic approximation method,” *The annals of mathematical statistics*, pp. 400–407, 1951.
- [92] T. Zhang, “Solving large scale linear prediction problems using stochastic gradient descent algorithms,” in *Proceedings of the Twenty-First International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 116. [Online]. Available: <https://doi.org/10.1145/1015330.1015332>
- [93] H. J. Kelley, “Gradient theory of optimal flight paths,” *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.
- [94] A. E. Bryson, “A gradient method for optimizing multi-stage allocation processes,” in *Proc. Harvard Univ. Symposium on digital computers and their applications*, vol. 72, 1961.
- [95] S. Dreyfus, “The numerical solution of variational problems,” *Journal of Mathematical Analysis and Applications*, vol. 5, no. 1, pp. 30–45, 1962.
- [96] A. E. Bryson, *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.
- [97] S. Linnainmaa, “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors,” *Master’s Thesis (in Finnish), Univ. Helsinki*, pp. 6–7, 1970.
- [98] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [99] —, “Learning internal representations by error propagation,” California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.
- [100] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.
- [101] Y. Nesterov, “A method for unconstrained convex minimization problem with the rate of convergence  $O(1/k^2)$ ,” vol. 269, pp. 543–547, 1983.
- [102] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *J. Mach. Learn. Res.*, vol. 12, no. null, p. 2121–2159, Jul. 2011.
- [103] M. D. Zeiler, “Adadelta: an adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012, tried to find an official citation but couldn’t find any.

- [104] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” pp. 26–31, 2012.
- [105] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: <http://arxiv.org/abs/1412.6980>
- [106] T. Dozat, “Incorporating nesterov momentum into adam,” in *International Conference on Learning Representations (ICLR) Workshop*. ICLR, 2016.
- [107] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of adam and beyond,” in *International Conference on Learning Representations*, 2018. [Online]. Available: <https://openreview.net/forum?id=ryQu7f-RZ>
- [108] L. Boltzmann, “On the relationship between the second fundamental theorem of the mechanical theory of heat and probability calculations regarding the conditions for thermal equilibrium,” *Entropy*, vol. 17, no. 4, pp. 1971–2009, 2015.
- [109] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [110] R. Salakhutdinov and G. Hinton, “Deep boltzmann machines,” in *Artificial intelligence and statistics*, 2009, pp. 448–455.
- [111] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [112] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. [Online]. Available: <https://doi.org/10.1162/neco.1989.1.4.541>
- [113] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, *Handwritten Digit Recognition with a Back-Propagation Network*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, p. 396–404.
- [114] Y. LeCun, C. Cortes, and C. Burges, “Mnist handwritten digit database.” [Online]. Available: <http://yann.lecun.com/exdb/mnist>
- [115] A. Bain, *Mind and body: The theories of their relation*. Henry S. King, 1873, vol. 4.

- [116] T. Kohonen, “Self-organized formation of topologically correct feature maps,” *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.
- [117] S. Hochreiter, “Untersuchungen zu dynamischen neuronalen netzen,” 1991.
- [118] J. F. Kolen and S. C. Kremer, *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, 2001, pp. 237–243.
- [119] S. Santurkar, D. Tsipras, A. Ilyas, and A. Mądry, “How does batch normalization help optimization?” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS’18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 2488–2498.
- [120] J. Kohler, H. Daneshmand, A. Lucchi, T. Hofmann, M. Zhou, and K. Neymeyr, “Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization,” in *Proceedings of Machine Learning Research*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and M. Sugiyama, Eds., vol. 89. PMLR, 16–18 Apr 2019, pp. 806–815. [Online]. Available: <http://proceedings.mlr.press/v89/kohler19a.html>
- [121] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *European conference on computer vision*. Springer, 2016, pp. 630–645.
- [122] R. K. Srivastava, K. Greff, and J. u. r. Schmidhuber, “Training very deep networks,” in *Advances in neural information processing systems*, 2015, pp. 2377–2385.
- [123] A. M. Thomson, “Neocortical layer 6, a review,” *Frontiers in neuroanatomy*, vol. 4, p. 13, 2010.
- [124] J. Winterer, N. Maier, C. Wozny, P. Beed, J. o. r. Breustedt, R. Evangelista, Y. Peng, T. D’Albis, R. Kempter, and D. Schmitz, “Excitatory microcircuits within superficial layers of the medial entorhinal cortex,” *Cell Reports*, vol. 19, no. 6, pp. 1110–1116, 2017.
- [125] D. Fitzpatrick, “The functional organization of local circuits in visual cortex: insights from the study of tree shrew striate cortex,” *Cerebral cortex*, vol. 6, no. 3, pp. 329–341, 1996.
- [126] R. K. Srivastava, K. Greff, and J. u. r. Schmidhuber, “Highway networks,” *CoRR*, vol. abs/1505.00387, 2015.
- [127] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

- [128] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, “Accurate, large minibatch sgd: Training imagenet in 1 hour,” *arXiv preprint arXiv:1706.02677*, 2017.
- [129] Y. You, Z. Zhang, J. Demmel, K. Keutzer, and C.-J. Hsieh, “Imagenet training in 24 minutes,” *arXiv preprint arXiv:1709.05011*, 2017.
- [130] T. Akiba, S. Suzuki, and K. Fukuda, “Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes,” *arXiv preprint arXiv:1711.04325*, 2017.
- [131] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu *et al.*, “Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes,” *arXiv preprint arXiv:1807.11205*, 2018.
- [132] H. Mikami, H. Suganuma, P. U.-Chupala, Y. Tanaka, and Y. Kageyama, “Imagenet/resnet-50 training in 224 seconds,” *CoRR*, vol. abs/1811.05233, 2018, nov 2018, 224 secs (3:44)- 2176 Tesla V100 - SONY. [Online]. Available: <http://arxiv.org/abs/1811.05233>
- [133] M. Yamazaki, A. Kasagi, A. Tabuchi, T. Honda, M. Miwa, N. Fukumoto, T. Tabaru, A. Ike, and K. Nakashima, “Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds,” *arXiv preprint arXiv:1903.12650*, 2019, march 2019, 84 secs - 2024 Tesla V100 - FUJITSU.
- [134] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.
- [135] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.
- [136] H. v. Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI’16. AAAI Press, 2016, p. 2094–2100.
- [137] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in *International Conference on Machine Learning*, 2016, pp. 1995–2003.
- [138] W. contributors, “Go (game) — wikipedia, the free encyclopedia,” [https://en.wikipedia.org/w/index.php?title=Go\\_\(game\)&oldid=934256256](https://en.wikipedia.org/w/index.php?title=Go_(game)&oldid=934256256), 2020, online; accessed 21-January-2020.

- [139] G. Lample and D. S. Chaplot, “Playing fps games with deep reinforcement learning,” in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [140] W. contributors, “Doom (1993 video game) — wikipedia, the free encyclopedia,” [https://en.wikipedia.org/w/index.php?title=Doom\\_\(1993\\_video\\_game\)&oldid=936771042](https://en.wikipedia.org/w/index.php?title=Doom_(1993_video_game)&oldid=936771042), 2020, online; accessed 21-January-2020.
- [141] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.
- [142] W. R. Thompson, “On the likelihood that one unknown probability exceeds another in view of the evidence of two samples,” *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.
- [143] J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate, “A bayesian sampling approach to exploration in reinforcement learning,” *arXiv preprint arXiv:1205.2664*, 2012.
- [144] T. Glasmachers, *Limits of End-to-End Learning*, 2017, vol. 77, p. 17–23.
- [145] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [146] W. Shi and S. Dustdar, “The promise of edge computing,” *Computer*, vol. 49, no. 5, pp. 78–81, 2016.
- [147] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
- [148] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile edge computing: A survey,” *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.
- [149] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.
- [150] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
- [151] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, “Mixed precision training,” *arXiv preprint arXiv:1710.03740*, 2017.

- [152] D. Das, N. Mellempudi, D. Mudigere, D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas *et al.*, “Mixed precision training of convolutional neural networks using integer operations,” *arXiv preprint arXiv:1802.00930*, 2018.
- [153] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, “Nvidia tensor core programmability, performance & precision,” in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 522–531.
- [154] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, “Mixed precision quantization of convnets via differentiable neural architecture search,” *arXiv preprint arXiv:1812.00090*, 2018.
- [155] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [156] W. contributors, “Cairo (graphics) — wikipedia, the free encyclopedia,” 2020, online; accessed 3-February-2020. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Cairo\\_\(graphics\)&oldid=936626191](https://en.wikipedia.org/w/index.php?title=Cairo_(graphics)&oldid=936626191)
- [157] Wikipedia contributors, “Opencv — Wikipedia, the free encyclopedia,” 2020, [Online; accessed 3-February-2020].
- [158] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
- [159] M. Weber, “Autotrace-converts bitmap to vector graphics,” 2004.
- [160] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results,” <http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html>.
- [161] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, “Panoptic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 9404–9413.
- [162] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.
- [163] T. Chen, X. Zhai, M. Ritter, M. Lucic, and N. Houlsby, “Self-supervised gans via auxiliary rotation loss,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 154–12 163.



- [164] M. Drozdal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, “The importance of skip connections in biomedical image segmentation,” in *Deep Learning and Data Labeling for Medical Applications*. Springer, 2016, pp. 179–187.
- [165] S. Jégou, M. Drozdal, D. Vazquez, A. Romero, and Y. Bengio, “The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 11–19.
- [166] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, “Icnet for real-time semantic segmentation on high-resolution images,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 405–420.
- [167] J. Li, W. Speier, K. C. Ho, K. V. Sarma, A. Gertych, B. S. Knudsen, and C. W. Arnold, “An em-based semi-supervised deep learning approach for semantic segmentation of histopathological images from radical prostatectomies,” *Computerized Medical Imaging and Graphics*, vol. 69, pp. 125–133, 2018.
- [168] S. Tokui, K. Oono, S. Hido, and J. Clayton, “Chainer: a next-generation open source framework for deep learning,” in *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, vol. 5, 2015, pp. 1–6.
- [169] Y. A. LeCun, L. e. o. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.
- [170] I. Loshchilov and F. Hutter, “Fixing weight decay regularization in adam,” in *Proceedings of the ICLR 2018 Conference*, vol. 30, 2018.
- [171] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, “Labeled faces in the wild: A database for studying face recognition in unconstrained environments,” 2008.
- [172] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “The omniglot challenge: a 3-year progress report,” *Current Opinion in Behavioral Sciences*, vol. 29, pp. 97–104, 2019.
- [173] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, “Think you have solved question answering? try arc, the ai2 reasoning challenge,” *arXiv preprint arXiv:1803.05457*, 2018.
- [174] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, “Superglue: A stickier benchmark for general-purpose language under-

standing systems,” in *Advances in Neural Information Processing Systems*, 2019, pp. 3261–3275.

## List of academic achievements

<p>Articles in refereed journals</p>	<ul style="list-style-type: none"> <li>○ Anh H. Dang, Wataru Kameyama, "SvgAI - Training Methods Analysis of Artificial Intelligent Agent to use SVG Editor", ICACT Transactions on Advanced Communications Technology (TACT), Vol.7, Issue 6, pp.1159-1166, Nov. 2018.</li> <li>○ Anh H. Dang, Wataru Kameyama, "Robust Semantic Segmentation for Street Fashion Photos", ICACT Transactions on Advanced Communications Technology (TACT), Vol.8, Issue 6, pp.1248-1257, Nov. 2019.</li> <li>● Mohammad A. Almogbel, Anh H. Dang, Wataru Kameyama, "Cognitive Workload Detection from Raw EEG-Signals of Vehicle Driver using Deep Learning", ICACT Transactions on Advanced Communications Technology (TACT), Vol.7, Issue 6, pp.1167-1172, Nov. 2018.</li> </ul>
<p>Presentations at International conferences</p>	<ul style="list-style-type: none"> <li>● Anh H. Dang, Wataru Kameyama, "Semantic Segmentation of Fashion Photos using Light-Weight Asymmetric U-Net," 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE 2019), Oct.15-Oct.18, Osaka, Japan, pp.177-180.</li> <li>● Anh H. Dang, Wataru Kameyama, "SvgAI – Training Artificial Intelligent Agent to use SVG Editor", IEEE 20th International Conference on Advanced Communications Technology (ICACTION 2018), 2B-02, Feb. 11-14, Elysian Gangchon, GW, Korea. (received Outstanding Paper Award)</li> <li>● Mohammad A. Almogbel, Anh H. Dang, Wataru Kameyama, "EEG-signals based cognitive workload detection of vehicle driver using deep learning", IEEE 20th International Conference on Advanced Communications Technology (ICACTION 2018), 3B-04, Feb. 11-14, Elysian Gangchon, GW, Korea.</li> <li>● Anh H. Dang, Wataru Kameyama, "Boosted Classifier by Intensity Tests", SS2-05, 2014 International Workshop on Smart Info-Media Systems in Asia (SISA2014), Oct.8-Oct.10, 2014, Hotel Majestic Saigon, Ho Chi Minh City, Vietnam.</li> <li>● D. H. Anh, G. G. D. Nishantha, Y. Hayashida and P. Davar, "A Flash-based lecture recording system and its integration with LMS," 2010 The 12th International Conference on Advanced Communication Technology (ICACTION), Phoenix Park, 2010, pp. 1425-1429.</li> <li>● D. Pishva, G. G. D. Nishantha and H. A. Dang, "A survey on how Blackboard is assisting educational institutions around the world and the future trends," 2010 The 12th International Conference on Advanced Communication Technology (ICACTION), Phoenix Park, 2010, pp. 1539-1543.</li> <li>● Nishantha, G. G. D., Dang H. Anh, Davar Pishva, William B. Claster, and Yukuo Hayashida. "Towards Developing an Integrated Multimedia Framework for Enhanced e-Learning." In CSEDU (1), pp. 267-272. 2009.</li> <li>● G.G.D. Nishantha, D.H. Anh, D. Pishva, W.B. Claster (2009) MULTIMEDIA ENHANCED UBIQUITOUS LEARNING MANAGEMENT SYSTEM, INTED2009 Proceedings, pp. 937-947.</li> </ul>
<p>Presentations at domestic</p>	<ul style="list-style-type: none"> <li>● Anh H. Dang, Wataru Kameyama, "Fish Detection by LBP Cascade Classifier with Optimized Processing Pipeline", IPSJ SIG AVM, Vol.2013-AVM-82, No.9, 2012.</li> </ul>

academic meetings held by study groups	
Presentations at domestic conferences	<ul style="list-style-type: none"> <li>• Anh H. Dang, Wataru Kameyama, “Image and HTML Document Cross Modal Analysis with CNN and LSTM Network”, IEICE General Conference, D-11-33, 2017.</li> <li>• Anh H. Dang, Pao Sriprasertsuk, Wataru Kameyama, “Scale Filtering for SURF-based AR Application”, IPSJ FIT, K-007, 2012.</li> <li>• Anh H. Dang, Pao Sriprasertsuk, Wataru Kameyama, “Task level SURF keypoint extraction parallelization in AR application”, IPSJ FIT, K-054, 2011.</li> </ul>
Published Books	<ul style="list-style-type: none"> <li>• Contributed to book chapter: “A Multimedia Integrated Framework for Learning Management Systems in the book "eLearning - Theories, Design, Software and Applications", Nishantha G.G.D., Anh H. Dang, et al. ISBN 978-953-51-0475-9 (INTECH Open Book Publisher 2012).</li> </ul>